***Forrest Bentley***

***B-Edit***

***User’s Guide and Language Reference***

**Forrest Benltey**

**612-388-3176**

**10/10/2013**

Forrest Bentley

B-Edit is a very powerful string manipulation language designed for batch editing text files. B-Edit is an interpretive language which processes B-Edit “script” files containing text editing commands. B-Edit can run batch in a Command Prompt window, as a Windows application, or interactively in a Command Prompt window.

B-Edit reads files into memory which gives it powerful file maneuvering abilities over conventional languages, such as easily reading and writing files, moving upward or downward in files, and using its many strong string manipulation functions you can produce the desired output text file. B-Edit is very useful when editing file rules are known and must be repeated.

B-Edit User’s Guide and Language Reference

Contents

[B-Edit User’s Guide 9](#_Toc416863101)

[Description 9](#_Toc416863102)

[History of B-Edit 9](#_Toc416863103)

[Installation 10](#_Toc416863104)

[Startup 10](#_Toc416863105)

[Programming Examples 11](#_Toc416863106)

[Example 1 11](#_Toc416863107)

[Example 2 14](#_Toc416863108)

[Example 3 17](#_Toc416863109)

[Recursion 18](#_Toc416863110)

[B-Edit Commands Overview 20](#_Toc416863111)

[File I/O 20](#_Toc416863112)

[Program Flow 20](#_Toc416863113)

[Variable Functions 20](#_Toc416863114)

[Screen I/O 21](#_Toc416863115)

[Editing 21](#_Toc416863116)

[BUFFER Commands 22](#_Toc416863117)

[SPECIAL Commands 22](#_Toc416863118)

[B-EDIT Language Reference 24](#_Toc416863119)

[Performance Notes 24](#_Toc416863120)

[Maintenance/Readability 25](#_Toc416863121)

[Special Notes 26](#_Toc416863122)

[B-Edit System Variables 27](#_Toc416863123)

[B-Edit Functions by Type 28](#_Toc416863124)

[File I-O 28](#_Toc416863125)

[Program Flow 29](#_Toc416863126)

[Variable Functions 31](#_Toc416863127)

[Screen I/O 33](#_Toc416863128)

[Editing 34](#_Toc416863129)

[Buffer 36](#_Toc416863130)

[Special 36](#_Toc416863131)

[Note 37](#_Toc416863132)

[B-EDIT Function List 38](#_Toc416863133)

[-number 38](#_Toc416863134)

[!LABEL 38](#_Toc416863135)

[$NAME 38](#_Toc416863136)

[\*ENDDATA 39](#_Toc416863137)

[@ 39](#_Toc416863138)

[ABS 40](#_Toc416863139)

[ACOS 40](#_Toc416863140)

[ARG 40](#_Toc416863141)

[ASIN 41](#_Toc416863142)

[ATAN 41](#_Toc416863143)

[B 42](#_Toc416863144)

[BACK 42](#_Toc416863145)

[BEDIT 42](#_Toc416863146)

[BEDITDATA 43](#_Toc416863147)

[BEGINEND 43](#_Toc416863148)

[BELL 43](#_Toc416863149)

[BOOKMARK 43](#_Toc416863150)

[BREAK 44](#_Toc416863151)

[BUFFER 45](#_Toc416863152)

[CALL 45](#_Toc416863153)

[CASE 48](#_Toc416863154)

[CHAIN 48](#_Toc416863155)

[CHAR 48](#_Toc416863156)

[CLEANKEYS 49](#_Toc416863157)

[CLEAR 49](#_Toc416863158)

[CLOSE 49](#_Toc416863159)

[CLS 49](#_Toc416863160)

[COLOR 50](#_Toc416863161)

[COLS 51](#_Toc416863162)

[COPY 51](#_Toc416863163)

[COS 52](#_Toc416863164)

[CRUSH 52](#_Toc416863165)

[CURRDIR 52](#_Toc416863166)

[CURSOR 53](#_Toc416863167)

[CUT 54](#_Toc416863168)

[D 54](#_Toc416863169)

[DEBUG 55](#_Toc416863170)

[DECRYPT 55](#_Toc416863171)

[DECRYPTFILE 55](#_Toc416863172)

[DIR 55](#_Toc416863173)

[DIRECTION 56](#_Toc416863174)

[DO 56](#_Toc416863175)

[DOLLAR 57](#_Toc416863176)

[DUMP 58](#_Toc416863177)

[ELSE 58](#_Toc416863178)

[ENCRYPT 59](#_Toc416863179)

[ENCRYPTFILE 59](#_Toc416863180)

[ENCRYPTKEY 59](#_Toc416863181)

[ENDIF 60](#_Toc416863182)

[EOL 60](#_Toc416863183)

[ERASE 60](#_Toc416863184)

[ERROR 61](#_Toc416863185)

[EXIT 61](#_Toc416863186)

[EXTRACT 61](#_Toc416863187)

[F 62](#_Toc416863188)

[FILE 62](#_Toc416863189)

[FILES 62](#_Toc416863190)

[FILESIZE 62](#_Toc416863191)

[FILESTRING 63](#_Toc416863192)

[FIXFILE 63](#_Toc416863193)

[FLOAT 64](#_Toc416863194)

[FRONT 64](#_Toc416863195)

[FS 64](#_Toc416863196)

[GOTO 65](#_Toc416863197)

[I 65](#_Toc416863198)

[IB 66](#_Toc416863199)

[IF 66](#_Toc416863200)

[INITIALIZE 70](#_Toc416863201)

[INPUT 70](#_Toc416863202)

[INPUTC 71](#_Toc416863203)

[INPUTP 71](#_Toc416863204)

[INTEGER 72](#_Toc416863205)

[INT 72](#_Toc416863206)

[LIST 72](#_Toc416863207)

[LEFTJUST 73](#_Toc416863208)

[LEN 73](#_Toc416863209)

[LOCAL 74](#_Toc416863210)

[LOGFILE 74](#_Toc416863211)

[LOG 74](#_Toc416863212)

[LOWER 75](#_Toc416863213)

[MD 75](#_Toc416863214)

[MIX 75](#_Toc416863215)

[name (variable name) 76](#_Toc416863216)

[NOCO 78](#_Toc416863217)

[NOTE 78](#_Toc416863218)

[NOVE 78](#_Toc416863219)

[OPEN 79](#_Toc416863220)

[P 79](#_Toc416863221)

[PACK 79](#_Toc416863222)

[PASTE 80](#_Toc416863223)

[PASTEB 80](#_Toc416863224)

[PATTERN 80](#_Toc416863225)

[PERM 81](#_Toc416863226)

[PRINT 81](#_Toc416863227)

[PRINTX 82](#_Toc416863228)

[PS 82](#_Toc416863229)

[R 83](#_Toc416863230)

[RANDOM 83](#_Toc416863231)

[RD 84](#_Toc416863232)

[RESTORE 84](#_Toc416863233)

[RETURN 85](#_Toc416863234)

[RIGHTJUST 85](#_Toc416863235)

[ROUND 86](#_Toc416863236)

[RS 86](#_Toc416863237)

[RV 86](#_Toc416863238)

[RVS 87](#_Toc416863239)

[SAVE 87](#_Toc416863240)

[SCAN 88](#_Toc416863241)

[SCANR 88](#_Toc416863242)

[SCRAMBLE 89](#_Toc416863243)

[SIN 89](#_Toc416863244)

[SIZE 89](#_Toc416863245)

[SLEEP 90](#_Toc416863246)

[SORT 90](#_Toc416863247)

[SPLIT 90](#_Toc416863248)

[SPLITR 91](#_Toc416863249)

[SQRT 91](#_Toc416863250)

[STATUS 91](#_Toc416863251)

[STRING 92](#_Toc416863252)

[SUBS 92](#_Toc416863253)

[SYSTEM 92](#_Toc416863254)

[SYSTEMCURR 93](#_Toc416863255)

[TAN 93](#_Toc416863256)

[TRACE 93](#_Toc416863257)

[UPPER 94](#_Toc416863258)

[UNTIL 94](#_Toc416863259)

[VERI 95](#_Toc416863260)

[WAIT 95](#_Toc416863261)

[WAITBELL 96](#_Toc416863262)

[Window 97](#_Toc416863263)

[WINDOW ACTION 104](#_Toc416863264)

[WINDOW AUTOTAB 104](#_Toc416863265)

[WINDOW BACKCOLOR 104](#_Toc416863266)

[WINDOW CENTER 105](#_Toc416863267)

[WINDOW CENTERDESKTOP 105](#_Toc416863268)

[WINDOW CHECK 105](#_Toc416863269)

[WINDOW CLEAR 105](#_Toc416863270)

[WINDOW CLOSE 105](#_Toc416863271)

[WINDOW CLOSEWINDOWNAME 105](#_Toc416863272)

[WINDOW COLOR 106](#_Toc416863273)

[WINDOW CONCEAL 106](#_Toc416863274)

[WINDOW CREATECONTROL 107](#_Toc416863275)

[WINDOW CREATEWINDOW 108](#_Toc416863276)

[WINDOW DELETE 108](#_Toc416863277)

[WINDOW DELETECONTROL 108](#_Toc416863278)

[WINDOW DELETEWINDOW 108](#_Toc416863279)

[WINDOW DEFAULT 108](#_Toc416863280)

[WINDOW DISABLE 109](#_Toc416863281)

[WINDOW EDIT 109](#_Toc416863282)

[WINDOW ENABLE 110](#_Toc416863283)

[WINDOW EXISTS 110](#_Toc416863284)

[WINDOW FILEINSERT 110](#_Toc416863285)

[WINDOW FOCUSWINDOWNAME 112](#_Toc416863286)

[WINDOW FONT 112](#_Toc416863287)

[WINDOW GETFOCUS 113](#_Toc416863288)

[WINDOW GETFOCUSWIN 113](#_Toc416863289)

[WINDOW HIDE 113](#_Toc416863290)

[WINDOW HWND 113](#_Toc416863291)

[WINDOW INSERT 114](#_Toc416863292)

[WINDOW LISTFONT 114](#_Toc416863293)

[WINDOW MASK 114](#_Toc416863294)

[WINDOW MESSAGE 115](#_Toc416863295)

[WINDOW MODIFY 116](#_Toc416863296)

[WINDOW MOUSEPOS 116](#_Toc416863297)

[WINDOW OBTAIN 116](#_Toc416863298)

[WINDOW OPEN 117](#_Toc416863299)

[WINDOW PAINT 118](#_Toc416863300)

[WINDOW PaintFile 118](#_Toc416863301)

[WINDOW POINTER 120](#_Toc416863302)

[WINDOW PUSH 120](#_Toc416863303)

[WINDOW SELECT 121](#_Toc416863304)

[WINDOW SELECTACTION 121](#_Toc416863305)

[WINDOW SETFOCUS 121](#_Toc416863306)

[WINDOW SHADE 122](#_Toc416863307)

[WINDOW SHOW 122](#_Toc416863308)

[WINDOW SIZE 122](#_Toc416863309)

[WINDOW SOUND 122](#_Toc416863310)

[WINDOW START 122](#_Toc416863311)

[WINDOW STOP 123](#_Toc416863312)

[WINDOW TIMER 123](#_Toc416863313)

[WINDOW TITLE 123](#_Toc416863314)

[WINDOW UNCHECK 124](#_Toc416863315)

[WINDOW UNCONCEAL 124](#_Toc416863316)

[WINDOW UNSELECT objname 124](#_Toc416863317)

[WINDOW UNSHADE 124](#_Toc416863318)

[WINDOW WINDOWPOS 124](#_Toc416863319)

[WRITE 126](#_Toc416863320)

# B-Edit User’s Guide

## Description

B-Edit is a very powerful string manipulation language designed for batch editing text files. B-Edit is an interpretive language which processes B-Edit “script” files containing text editing commands. B-Edit can run batch in a Command Prompt window, as a Windows application, or interactively in a Command Prompt window. B-Edit reads files into memory which gives it powerful file maneuvering abilities over conventional languages, such as easily reading and writing files, moving upward or downward in files, and using its many strong string manipulation functions you can produce the desired output text file. B-Edit is very useful when editing file rules are known and must be repeated. PC memory capacity is the only limiting factor to the size or number of text files which can be processed concurrently which means very large files can be processed using current PC hardware. E.g. If you had a need to update all program files in a directory with a new copyright notice (or any other text), you could use B-Edit to open each file, insert the copyright, and write the updated files, very easily.

## History of B-Edit

GE Fleet was originally named GELCO (General Equipment Leasing Company) and was purchased by GE around 1987. At that time GELCO was running all of its systems on a Honeywell mainframe. Honeywell had a system called TEX (Text Executive Processor) which allowed users to develop script files to batch edit multiple files simultaneously to produce the desired output file(s). IBM didn’t have a similar product, and developers at GELCO had a number of very useful scripts written in TEX that were going to be lost after converting to the IBM mainframe, so I decided to write an interpreter in C++ that did everything TEX did, plus many enhancements, and called this new language B-Edit. It’s a very useful language for batch editing files, and I’ve used it extensively during my GE career.

**Note: If I was designing a language without constraint it likely wouldn’t have the syntax of B-Edit, but since it was originally patterned after TEX it had to conform to the syntax of TEX to preserve the existing TEX scripts already written. I’ve enhanced B-Edit far beyond its original TEX beginnings, but the syntax still conforms to TEX.**

Here are some examples of some complex script programs I’ve written for B-Edit:

* Converting all CICS COBOL transactions to DC-COBOL for use via MQ (which included downloading the source from the mainframe, converting the code, uploading the converted code back to the mainframe, creating compile JCL).
* Scan code (Java, Javascript, HTML, C++, COBOL, ADSO, etc…) for GE standards, including Sigma ratings.
* Scan code for cross-reference data to load an Enterprise Architecture database which allowed impact analysis research
* Generating C++ and COBOL code for client server applications at GE Fleet (up to 90% of some systems at Fleet were generated using B-Edit).

## Installation

To install B-Edit copy BEDIT.EXE into a directory that’s in your path.

If you want to run the Windows version copy BEDITWIN.EXE into a directory that’s in your path.

To display your path go to a DOS prompt and enter “path”.

To edit your path click on the Windows icon in the lower left of your taskbar, click on “Computer”, click the “System properties” tab, click on “Advanced system settings”, click “Yes” if the User Account Control prompt appears, under the “Advanced” tab click the “Environment Variables” button, in the “System variables” list click on “path”, click “Edit” and make your changes separating directories with a ‘;’. After you’re finished making changes click the “OK” button on the “Edit System Variable” screen, click “OK” on the “Environment Variables” screen, click “OK” on the “System Properties”screen, your Path changes will now be applied. Make sure to close all open DOS windows so the next time you open one the Path changes will be in use.

## Startup

To start B-Edit in interactive mode enter “BEDIT” under a DOS prompt.

This will place you at the B-EDIT> prompt where you may enter any B-Edit command

e.g. Enter “print:’Hello World’” and press enter.

A B-Edit script file contains a set of B-Edit commands that will be executed in batch mode, including program flow commands such as “CALL”, IF-ELSE, DO-WHILE, etc… To run a B-Edit script file you can enter “CALL script-filename” at the >B-EDIT prompt, or you could create a shortcut on your desktop, or in a desktop folder, with the following properties:

BEDIT.EXE CALL scriptfilename

This will launch the B-Edit interpreter executable which will read the “Called” script file into memory and begin processing commands.

## Programming Examples

### Example 1

Let's start with a simple B-Edit program. The goal of this program

will be to read in a Java file, print all occurrences of the word “public”, then change all occurrences of userid to user\_id, then add a revision log title area to the top of the program.

To achieve this goal we’ll create a B-Edit script file with the following B-Edit commands:

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* PROG1 : Read and edit a Java program

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

print:"PROG1 : This is a test B-Edit program"

open file1 c:\bedit\Prog1.java

b ps:/public/;\*

b rvs:/userid/:/user\_id/;\*

b fs:”public”

ib:’/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*’

i:’ \* Revision Log (mm/dd/ccyy) Name Description’

i:’ \* -----------------------------------------------------------------------------------------‘

i:’ \*/’

write file1 c:\bedit\Prog1b.java

close file1

print:"PROG1 : Processing complete"

wait

exit

After you have created this save it as “PROG1.BED”.

To run the script enter “bedit call prog1.bed” at a DOS prompt in the directory containing prog1.bed, or create a shortcut on your desktop (or desktop folder) with the following properties “Bedit.exe call prog1.bed” starting in the directory that contains prog1.bed. You’ll need a file called Prog1.java in the same directory.

When you run this script B-Edit will display all of the lines in Prog1.java that contain “public”, then you’ll see all of the lines that contained “userid” changed to “user-id”, then you’ll be prompted to hit any key due to the “wait” command, then the script will be complete. After the script has been run Prog1b.java will be created with all of the changes, including the Revision log title that you entered in the script. In this example you’ll notice that multiple commands can be entered on one script line which is fine. If you do this remember to keep the script understandable.

Example Details:

Lines 1-3 are comment lines. Double slashes anywhere on a line indicate that from that point to the end of the line will be considered comments.

Line 4) print:"PROG1 : This is a test B-Edit program"

Instructs B-Edit to output text onto the DOS screen.

Line 5) open file1 c:\bedit\Prog1.java

Instructs B-Edit to open the file c:\bediit\Prog1.java and load it into an internal memory file named “file1” . Any future reference to this file will be done using the internal filename “file1”. Note: The entire file will be loaded into memory for processing.

Line 6) b ps:/public/;\*

The “b” will move the file pointer to the beginning of the file (first line)

The “ps” will find and print the string “public”, and the trailing “;\*” indicates to repeat the “ps” command to the end of the file.

Line 7) b rvs:/userid/:/user\_id/;\*

The “b” will move the file pointer to the beginning of the file (first line).

The “rvs” change and print (reverse & verify) any occurrence of “userid” to “user\_id”, and the trailing “;\*” indicates to repeat the “rvs” command to the end of the file.

Line 8) b fs:”public”

The “b” will move the file pointer to the beginning of the file (first line).

The “fs” will find the first occurrence of the string “public”.

Note: Notice that you can use any delimiter characters.

Line 9) ib:’/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*’

The “ib” will insert the text characters before the current file position. Remember in the last command we positioned the file pointer to the first line containing “public”, so this new line will be inserted before “public”.

Line 10-11) i:’ \* Revision Log (mm/dd/ccyy) Name Description’

i:’ \* -----------------------------------------------------------------------------------------‘

These two lines will insert new text lines after the current position, which at this point is the new line inserted with the “ib” command.

Line 12) write file1 c:\bedit\Prog1b.java

This command will write the memory file “file1” (that we’ve been editing) to c:\bedit\Prog1b.java.

Line 13) close file1

This command will remove file1 from memory so it’s no longer taking space and can’t be referenced again.

Line 14) print:"PROG1 : Processing complete"

This program will output the text on the DOS screen.

Line 15) wait

This command will prompt you to press any key before continuing.

Line 16) exit

This command will exit the B-Edit interpreter.

### Example 2

A more useful program may read the same source, write all lines that contain the word “public “ into another file. Doing this would allow another program to interrogate the “public” lines and produce a list of all publicly available classes.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* PROG2 : Read a Java program and write out public lines

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

print:"PROG2 : This is a test B-Edit program"

string PublicLine

open Java c:\bedit\Prog1.java

open Public NULL

file Java

b

call !A100\_Public

write Public c:\bedit\public.dat

exit

!A100\_Public

fs:/public/

if (EOF) goto !A100\_Return

PublicLine = @line@

file Public

i:/@PublicLine@/

file Java

f goto !A100\_Public

!A100\_Return

return

Line 1-3) Again are comment lines.

Line 4) print:"PROG2 : This is a test B-Edit program"

This line will print text on the DOS screen.

Line 5) string PublicLine

This line will create a new internal string variable name “PublicLine”.

Line 6) open Java c:\bedit\Prog1.java

This line will open the file c:\bedit\Prog1.java and place its contents in a memory file named “Java”.

Line 7) open Public NULL

This line will create a new internal file named “Public” without any contents.

Line 8) file Java

This line will change the current file pointer to the “Java” file. We need to do this since in the prior line we just create a new file “Public” which was the current file pointed to.

Line 9) b

This line will move the current line pointer (in file “Java”) to the top of the file.

Line 10) call !A100\_Public

This line will call a subroutine with the name “!A100\_Public”. Once this subroutine is complete it will return to the next command following the call (Note: the next command could be on the same line).

Line 11) write Public c:\bedit\public.dat

This line will write the memory file “Public” to the output file location c:\bedit\public.dat

Line 12) exit

This command will exit the B-Edit interpreter.

Line 13) !A100\_Public

This line indicates a Label named “!A100\_Public, which in this case is the start of the subroutine “!A100\_Public”. Subroutines can be named anything, but the “!” character must start in position 1 of the line. Processing will continue within the subroutine until a “return” command is encountered (or an “exit”).

Line 14) fs:/public/

This line will find the string “public” in the current file (which is currently the file “Java”).

Line 15) if (EOF) goto !A100\_Return

This line will check to see if we hit the end-of-file (EOF) so no more lines containing “public” were found. If we are at the end-of-file then control is passed to a label called !A100\_Return. Goto’s are not the preferred programming style, but are useful when common processing is needed at the end of a subroutine, so use them carefully to keep your code understandable.

Line 16) PublicLine = @line@

This line will copy the contents of the current line (currently a line in “Java” that contains the text “public”) into the string variable “PublicLine”. The “@” character instructs B-Edit to look for the variable “line” which is the line pointed to in the current file, and use the contents of “line” for the assignment.

Line 17) file Public

This line will change the current file pointer to the memory file “Public”

Line 18) i:/@PublicLine@/

This line will insert the contents of the string variable “PublicLine” into the current file, which we just changed to the file “Public”

Line 19) file Java

This line will change the current file pointer back to the file “Java” so we can search for more This command will exit the B-Edit interpreter lines containing “public”.

Line 20) f goto !A100\_Public

The ‘f’ command will more the current line position forward 1 line, and the goto will pass control to the Label !A100\_Public.

Line 21) !A100\_Return

This line defines the label “!A100\_Return”.

Line 22) return

This line will return control back to the first command following the “Call” that initiated the subroutine.

### Example 3

There are many ways to accomplish the same task, here’s a simpler program to accomplish the same task that illustrates more functions within B-Edit:

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* PROG3 : Read a Java program and write out public lines

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

print:"Prog3 : This is a test B-Edit program"

string Filename

printx:"Please enter the Java filename:"

input Filename

open Java @Filename

b

copy:/public/;\*

open Public null

paste

write Public Public.dat

exit

In this example the CUT/COPY buffer is used to hold the COPY'ed PERFORM lines. After all lines have been copied the current file is switched to "Performs" and the contents of the buffer is PASTE'd.

Line 10 uses a version of COPY that looks for a string, and in this case the “\*” looks for all occurrences from the current file position to the end of the file. Copy also allows you to copy lines a number of times without looking for a string within the line.

Line 12 uses PASTE to move all of the contents of what’s currently in the Copy Buffer into the current file which at this point in the program is the file named “Public”.

## Recursion

One of the more powerful features of B-Edit is its recursive ability within variable names. In the previous examples the use of @line@ was used to assign the contents of the variable "line" to another variable. If the contents of a variable used in this manner contains another @name@ then the contents of the variable that is pointed to will be used. Use the @^ to print the contents of the pointed to variable.

i.e.:

string var1

string var1ptr

var1 = "This is a test"

var1ptr = "@var1"

print:”@^var1ptr@”

In this example "This is a test" will be printed. The ‘@^’ causes the contents of var1ptr (@var1) and since the contents start with ‘@’ B-Edit will follow the pointer chain to var1 which contains ‘This is a test’, so continuing the chain the contents of var1 are Printed. This recursive capability is unlimited, allowing unlimited levels of pointers (be careful not to make code too confusing).

This is very useful when creating dynamic code. i.e. If you wanted to create a table of strings you could use the following code :

integer count

string test

count = 0

!A100\_Loop

count = count + 1

if (count > 10) goto !A100\_Continue

test = "test" + count

string @test

goto !A100\_Loop

!A100\_Continue

When control is passed to !A100\_Continue, 10 new string variables have been created (test1, test2, test3... test10).

This Functionality is consistent throughout B-Edit. Wherever a command uses a variable name, you may use the @name to use the contents of a variable, and if the contents of the variable contain an @name then the contents of that variable will be used, etc...

You may want to have the last variable in the chain of @variables to point to a variable without changing the data to include the @. To do this final recursion pointing use @@variable. This works for assignment, but it will not work for printing since the @'s should enclose a variable name in the print.

i.e.

Varptr1 = "Var2"

string @Varptr1 = "FINAL" // create string Var2 = 'FINAL'

string Var3 = @@Varptr1 // create Var3 = 'FINAL'

## B-Edit Commands Overview

### File I/O

You have seen how OPEN, WRITE, and FILE work. CLOSE is used to clear memory after you have completed using a given file. If you are using B-Edit interactively you may want to use the FILES command to see what files you have currently open. FILESIZE allows you to change all lines in a file to a fixed length. ENCRYPTKEY, ENCRYPTFILE, DECRYPTFILE functions can be used to encrypt/decrypt entire files prior to I/O, including running an encrypted B-Edit script file using the –E: calling parameter (See CALL)

### Program Flow

You have seen how GOTO, CALL, RETURN, EXIT, and some of the IF statement. IF statements allow you to test for EOF, or EXISTS (if a PC file exists), if a variable equals another variable, if a variable doesn't equal another variable, if a variable is less than another variable, if a variable is greater than another variable.

EXIT will exit the B-Edit system, unless you have multiple levels of B-Edit running.

i.e. if you enter BEDIT and at the B-Edit prompt you enter BEDIT, entering EXIT will bring you back one level within B-Edit, back to the first B-EDIT prompt.

### Variable Functions

Use STRING to create a string variable or INTEGER to create an integer variable. You must create a variable before trying to use it. If you want to see the contents of a variable use the DUMP command. If you want to initialize a variable use INITIALIZE. If you want to remove a variable completely use the ERASE command. String variables may be added to, and integer variables may be added, subtracted, multiplied, and divided. Conversion between types is supported, i.e.

stringvar = stringvar + integervar

SCAN and SCANR functions are a very powerful way of working with strings. These functions allow you to scan a variable for a string and the contents of the left, middle, and right are moved into the special variables left, mid, right, and the variable found is set to 'Y' if the string is found, or 'N' if the string was not found. SCAN scans the variable from left to right while SCANR scans the variable from right to left.

SPLIT and SPLITR functions give you another powerful way to work with strings. These functions allow you to split up a variable at a certain point. Again the special variables left and right are updated appropriately. SPLIT splits starting from the left while SPLITR starts splitting from the right.

SIZE is a very useful function for changing the size of a variable.

PACK is a useful way of removing trailing spaces from a variable.

CRUSH is a useful way to remove all spaces from a variable.

RIGHTJUST is a useful way to right justify a variable.

LEFTJUST is a useful way to left justify a variable.

MIX is a useful way to Mix the Case of a variable, i.e. if var = "this is a test" then after mix var = "This Is A Test"

LOWERCASE is a useful way to change all characters to lower case.

UPPERCASE is a useful way to change all characters to upper case.

INPUT allows you to enter text from the keyboard into a variable.

CHAIN will show the entire recursive chain for a variable.

ENCRYPT will encrypt a variable

DECRYPT will decrypt a variable

ENCRYPTKEY will set the encryption key for ENCRYPT, DECRYPT, ENCRYPTFILE, DECRYPTFILE, B-Edit call

parameter –E:

### Screen I/O

PRINT allows you to print messages or variables to the screen with a carriage return line feed.

PRINTX allows you to print messages or variables to the screen without a carriage return line feed.

WAIT will pause until a key is pressed.

DUMP allows dumping variable contents.

CURSOR allows you to move the cursor anywhere on the screen.

COLOR allows you to change the character color printed in the display window.

### Editing

VERI turns verify on so all changes or movement within a file are printed to the screen.

NOVE turns verify off so not all changes or movement within a file are printed to the screen.

"I" puts you into insert mode. Entered text is inserted into the current file until you ENTER without any text.

B-Edit supports the use of all of the standard editing functions such as B, FS:/xxx/, PS:/xxx/, RVS:/xxx/:/yyy/, R:/xxx/, I:/xxx/, P, D, with many variations on each function.

You may use COLS to mask off certain columns for editing (turned off using NOCO).

You may use BEGINEND to find a line that begins, or ends with a string.

### BUFFER Commands

CUT removes lines from the current file and places them into the cut/copy buffer.

COPY copies lines from the current file and places them into the cut/copy buffer.

PASTE copies the cut/copy buffer into the current file.

BUFFER shows the contents of the cut/copy buffer.

CLEAR removes all lines from the cut/copy buffer.

Note: The cut/copy buffer is cleared before a CUT or COPY if a PASTE has just taken place.

### SPECIAL Commands

Note lines are defined using // or \\ or NOTE:"..." or \_

Everything on the line following one of the note definers will not be executed by B-Edit.

PATTERN allows you to select what characters will be used in pattern matching for FS, PS, RV, RVS commands.

BEDIT command will envoke another level of Bedit.

STATUS command shows the current status of certain B-Edit commands (COLS, FILES, CALL Level, B-Edit Level, Variable count, etc...).

SYSTEM command is a useful way to send commands to DOS.

SYSTEMCURR command is a useful way to process a system command in DOS and the results of the command will be placed in the current file.

The directory commands DIR, CD, MD, RD may be used directly.

SLEEP command allows you to pause processing.

DEBUG command allows you to debug your B-Edit Command files.

TRACE command allows you to debug your B-Edit Command files.

$name is a useful way of dynamically executing B-Edit commands.

i.e. if var = "b p;3 f;3 p;5" then $var will execute the commands within var.

Note : When in interactive mode using the cursor up/down keys will scroll through all previously entered commands.

# B-EDIT Language Reference

## Performance Notes

* B-Edit keeps track of the last label a line has transferred control to, and will try to use that label the next time the line transfers control. i.e. If the line reads 'goto !a100\_Label1' B-Edit will find where !a100\_Label1 the first time the line is executed. All subsequent times the line is executed B-Edit will not search from the top looking for the label since it was found the first time. If, however the line reads 'call !a100\_Test goto !a100\_Label' the top down search will always happen since B-Edit keeps track of only the last transfer label (First !a100\_Test is transferred to, and next !a100\_Label). Your code will run much faster if you don't have two program flow statements on one line.
* Keep the most used !Labels close to the top as this will improve processing speed.
* Try to use as few variables as possible (keep the routine readable though). When accessing variables B-Edit will begin searching for the variable at the top of the list and move downward. Define your most used variables first so they appear at the top of the search list. Erase variables when you have finished using them.
* Use LOCAL variables whenever possible. These variables will be released after the CALLed routine is complete.
* Put as many commands on one line as possible. This will speed up processing but it will also make the code more difficult to read. Try to strike a balance between the two.
* If a file will be CALLed many times make it PERManent. Keep in mind that a PERMenant routine will remain in memory until B-Edit is terminated.
* If possible split up your code into multiple files and make them PERMenant. This has a few performance impacts. Splitting the code will make the initial load time shorter. Also, when a file is CALLed only the !Labels within that file are searched when using CALL or GOTO. By splitting a program up you will be making the list of !Labels to search for shorter. This will have a great impact on performance.
* When processing a file, you may want to use the 'FS' command to find the line you want before processing. This will increase the speed of the routine. i.e. If you process a file line by line using SCAN or SPLIT on each line to find a string, the routine will run much faster if you FS the string and then do the SCAN or SPLIT to process the line.
* Keep your routines in a directory that is close to the start of the B-Edit search path (current directory, SET BEDIT=...;...;, SET DPATH=...;...;). This will keep the search time to a minimum.
* Use 'ERROR' processing when defining variables in routines that may be called by other routines. This will allow multiple CALL's to the routine. Otherwise release all variables and close all files when a routine is complete. i.e. string w100-test1 = "this is a test" error
* Use the variable 'calllevel' to determine how to exit a routine. If a routine is called directly from B-Edit (bedit call filename) the routine should probably EXIT when complete. If a routine is called by another routine it should probably RETURN when complete. Using calllevel allows a routine to determine how to finish.

i.e. if (calllevel > 1)

+ return

+ else

+ exit

+ endif

## Maintenance/Readability

* Use indentation appropriately to improve maintainability.

i.e. string S100-Test1 = "This is test1" error

string S100-Test2 = "This is another test" error

integer I200-Counter = 10 error

!A100\_Start

if (I200-Counter > 5)

+ print:"I200-Counter is greater than five"

+ else

+ if (S100-Test1 = S100-Test2)

+ print:"S100-Test1 = S100-Test2"

+ else

+ print:"S100-Test1 != S100-Test2"

+ endif

* Use Macro's appropriately to improve maintainability.

i.e. string RED = "color 1"

string NORMAL = "color default"

$NORMAL print:"This is " $RED print:"RED" $NORMAL print:"text!"

* Use comments and space lines to split up and document the program.
* Use Naming conventions to keep code consistant.

i.e. string S100-String1

integer I100-Integer1

!A100\_Label1

* Modularize your programs.

i.e. !A100\_Main

call !B100\_Init

call !B200\_Proc

call !B300\_Wrap

return

* Print messages to the user during long running processes to show what the routine is doing.
* Keep screen headings consistent.

## Special Notes

* B-Edit is case sensitive by default. This includes label names in CALL's and GOTO's, variable names, etc. If you use CASE OFF to turn the case sensitivity off then ALL checking within B-Edit is not case sensitive. Keep this in mind when naming labels, variables, etc.
* In most cases when referencing a variable using @name@ the @ following the name is optional. The case where the @ following the name is required is when using it in a string. i.e. print:"test1=[@test1@]" (This applies to PRINT, PRINTX, and variable assignment i.e. test = "@test1@ @test2@"). If you leave off the second @ then the string will be unchanged i.e. test = "@test" + count, if count = 1 then test will be "@test1". This allows you to still use variables dynamically.
* There are limitations set on B-Edit. STRING, VARIABLE, FILE names may only be 100 characters long. The maximum number of characters allowed in a STRING, FILE record, or any internal variable is 1000 characters. 1000 is default. If you want to change this number use the -number option when starting B-Edit. i.e. If you want to set the max characters to 2500 then enter:

bedit -2500

bedit -2500 call example.bed

* B-Edit is designed to handle text type data files only (Not EXE's, COM's, etc). You will get strange results if you try reading in on non-text type file! Keep these limitations in mind when using B-Edit.
* You should not use any reserved word (i.e. if, open, list, goto, etc...) for any user defined variable name.

## B-Edit System Variables

Line Contains the current file's current line text.

lline: Length of line

lineno Current file's current line number

found Contains 'Y' if a scan function found the string, otherwise 'N'.

left Contains the left side of the string after a scan or split.

lleft Length of left

mid Contains the scan for string after a successful scan.

lmid Length of mid

right Contains the right side of the string after a scan of split.

lright Length of right

midright Contains mid and right combined

lmidright Length of midright

lword Contains the number of spaces to the left of scan:/@var/:word

llword Length of lword

rword Contains the line starting with the first non blank in a scan:/@var/:word

lrword Length of rword

partial1 Internal variable used in IF var[x-y] logic

partial2 Internal variable used in IF var[x-y] logic

functionkey Contains the function key pressed if one was pressed when using the INPUT function

(F1, F2, ...F12, SHFT-F1, SHFT-F2, ...SHFT-F12, CTRL-F1, CTRL-F2, ...CTRL-F12).

If a function key was not pressed this variable will be null. 'functionkey' will contain the following values depending on the function key pressed terminating the input:

F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12

SHFT-F1, SHFT-F2, SHFT-F3, SHFT-F4, SHFT-F5, SHFT-F6,

SHFT-F7, SHFT-F8, SHFT-F9, SHFT-F10, SHFT-F11, SHFT-F12,

CTRL-F1, CTRL-F2, CTRL-F3, CTRL-F4, CTRL-F5, CTRL-F6,

CTRL-F7, CTRL-F8, CTRL-F9, CTRL-F10, CTRL-F11, CTRL-F12,

INSERT, DELETE, HOME, END, PAGEUP, PAGEDOWN,

UP, DOWN, TAB, BACKTAB, ESC, UP, DOWN, RIGHT, LEFT (LEFT only for INPUTC).

pcount Number of times a function was processed (b, p, f, fs, ps, d,rs, rvs, cut, copy)

calllevel Contains the level of files called. The first file CALLed (i.e. BEDIT CALL TEST.BED) will set the 'calllevel' variable to 1, If this routine CALLs another file the 'calllevel' will be set up by 1. Each RETURN from a CALLed file will set 'calllevel' down by 1. You may want to use this variable to see if a routine is being CALLed directly, or from another routine, allowing you to either EXIT or RETURN. i.e. if (calllevel > 1) return else exit

null Contains a null string.

Tab Contains a Tab character for tab checking, or inserting a tab into text.

date Contains the current date (mm/dd/yy)

time Contains the current time (hh:mm:ss)

## B-Edit Functions by Type

### File I-O

OPEN name filestring open @name@ @name@

open name null

READ name filestring read @name@ @name@

read name NULL

read @name@ NULL

WRITE name filestring write @name@ @name@

write \* filestring

write \* @filestring

write \* \*

SORT name,type,start sort @name@,type,@name@

sort \*,type,start

CLOSE name close @name@

FILESIZE name;start;end filesize @name;@start;@end

filesize \*;@start;var

FRONT:/.../;n front:/@name/;@name

BACK:/.../;n back:/@name/;@name

FILE name file @name@

FILES files

SYSTEMCURR:"..." systemcurr:"@name@"

BEDITDATA beditdata

... ...

\*ENDDATA \*enddata

### Program Flow

!Label

GOTO !Label goto @name@

CALL !Label call Filename

call @name@

call \*

call Filename!Label

Var = call !Label [Arg1...Argn]

RETURN return

EXIT exit

DO-UNTIL

IF (EOF) (eof)

(!EOF) (!eof)

(PM) (!pm)

(NOTEOF) (noteof)

(found) (!found)

(EXISTS Filename) (exists @name@)

(NOTEXISTS Filename) (notexists @name@)

(!EXISTS filename) (!exists @name@)

(KEY) (key)

(NOKEY) (nokey)

(!KEY) (!key)

(NUMERIC name) (!numeric name)

(DIGIT name) (!digit name)

(SPECIAL name) (!special name)

(ALPHA name) (!alpha name)

(name) (!name)

(name = name) @name@ all cases

(name = "...") (name = n)

(name != name) (name != "...")

(name != n) (name > name)

(name > "...") (name > n)

(name < name) (name < "...")

(name < n) (name <= name)

(name <= "...") (name <= n)

(name >= name) (name >= "...")

(name >= n)

(numeric name[x-y]) (!numeric name[x-y])

(name[x-y] = name2[x-y]) (name[x-y] = '...')

(numeric name[x,y]) (!numeric name[x,y])

(name[x,y] = name2[x,y]) (name[x,y] = '...')

(numeric name[x|y]) (!numeric name[x|y])

(name[x|y] = name2[x|y]) (name[x|y] = '...')

(name) or (name2) and (name3)

ELSE else

ENDIF endif

ERROR error

### Variable Functions

SUBS char subs char

SAVE SYSVARS save uservars

SAVE \* save name

RESTORE SYSVARS restore uservars

RESTORE \* restore name

LOCAL local

ARG arg

STRING name string @name@

STRING name = '...' string @name = '...'

INTEGER name integer @name@

INTEGER name = 1 integer @name = 2

DOLLAR name dollar @name@

DOLLAR name = 1.23 dollar @name = 2.34

FLOAT name float @name@

FLOAT name = 1.234 float @name = 2.345

ERASE name erase @name@

ERASE \* erase \*

ERASE UNSAVED erase unsaved

INITIALIZE name initialize @name@

INITIALIZE \* initialize \*

name = name + name (+ - \* /) @name@ = @name@ + @name@

DOLLAR fields give 2 decimal accuracy

name = name + n @name@ = @name@ + n

name = int1 + int2 / int3... @name = @int1 + @int2 / @int3 ...

name = name + "..." @name@ = @name@ + "..."

@name@ = @name@ + "@name1@ @name2@..."

@name@ = "@name[x-y]@..."

@name = "@name[@name-@name]@"

@name = "@name[end-@var]"

name += nameb @name@ += @nameb@

name -= nameb @name@ -= @nameb@

name \*= nameb @name@ \*= @nameb@

name /= nameb @name@ /= @nameb@

name = "..." @name@ = "..."

@name@ = "@name1@ @name2@..."

name = n @name@ = n

name = LEN(name) @name@ = len(@name@)

name ++ @name@ ++ (Note: Make sure there’s a space before the ++)

name -- @name@ -- (Note: Make sure there’s a space before the --)

name += nameb @name@ += @nameb@

name -= nameb @name@ -= @nameb@

name \*= nameb @name@ \*= @nameb@

name /= nameb @name@ /= @nameb@

SCAN:/xxx/:/xxx/ scan:/@name@/:/@name@/

scan:@name@:word

SCANR:/xxx/:/xxx/ scanr:/@name@/:/@name@/

scanr:@name@:word

SPLIT:/xxx/;n split:/@name@/;@name@

SPLITR:/xxx/;n splitr:/@name@/;@name@

EXTRACT name,start,length

extract @name@,@startname@,@lenname@

INPUT name input @name@

INPUT name,# input @name,@number

CHAIN name chain name

### Screen I/O

PRINT:"xxx" print:"@name1@ @name2@..."

PRINTX:"xxx" printx:"@name1@ @name2@..."

WAIT wait

DUMP name dump name

DUMP \* dump \*

LIST list n-n

list -n

list n-

list page

list next

list prev

list n-page

CURSOR #,# cursor @name,@name

CURSOR SAVE cursor save

CURSOR RESTORE cursor restore

CURSOR UP,# cursor up,@name

DOWN,# down,@name

RIGHT,# right,@name

LEFT,# left,@name

EOL eol

### Editing

FS,PS,RS,RVS use Pattern Matching (?=any, #=number, &=alpha, \*=end)

DIRECTION UP direction up

DIRECTION DOWN direction down

VERI veri

NOVE nove

COLS start,end cols @name@,@name@

NOCO noco

BEGINEND BEGIN beginend begin

BEGINEND END beginend end

BEGINEND OFF beginend off

B B;n B;\* b;@name@

F F;n F;\* f;@name@

FS:/xxx/ FS:/xxx/;n fs:/@name@/;/@name@/

FS:/xxx/+/yyy/;n fs:/@name@/+/@name@/;@name@

FS:/xxx/-/yyy/;n fs:/@name@/-/@name@/;@name@

PS:/xxx/ PS:/xxx/;n ps:/@name@/;/@name@/

PS:/xxx/+/yyy/;n ps:/@name@/+/@name@/;@name@

PS:/xxx/-/yyy/;n ps:/@name@/-/@name@/;@name@

RS:/xxx/:/yyy/ rs:/@name@/:/@name@/

RS:/xxx/:/yyy/;n rs:/@name@/:/@name@/;@name@

RS:/xxx/:/yyy/;\* rs:/@name@/:/@name@/;\*

RVS:/xxx/:/yyy/ rvs:/@name@/:/@name@/

RVS:/xxx/:/yyy/;n rvs:/@name@/:/@name@/;@name@

RVS:/xxx/:/yyy/;\* rvs:/@name@/:/@name@/;\*

R:/xxxxx/ r:/@name@/

RV:/xxxxx/ rv:/@name@/

I:/xxx/ i:/@name@/

I i

IB ib

IB:/xxx/ ib:/@name@/

P p

P;n p;@name@

P;\* p;\*

D d

D;n d;@name@

D;\* d;\*

### Buffer

CUT cut

CUT;n cut;@name@

CUT;\* cut;\*

CUT:/xxx/ FS:/xxx/;n cut:/@name@/;/@name@/

CUT:/xxx/+/yyy/;n cut:/@name@/+/@name@/;@name@

CUT:/xxx/-/yyy/;n cut:/@name@/-/@name@/;@name@

COPY copy

COPY;n copy;@name@

COPY;\* copy;\*

COPY:/xxx/ FS:/xxx/;n copy:/@name@/;/@name@/

COPY:/xxx/+/yyy/;n copy:/@name@/+/@name@/;@name@

COPY:/xxx/-/yyy/;n copy:/@name@/-/@name@/;@name@

CLEAR clear

PASTE paste

PASTEB buffer

BUFFER

### Special

NOTE:"..." note:"..."

//

\\

\_

PATTERN ON pattern on

PATTERN OFF pattern off

PATTERN ?#&~\* pattern xxxxx

BEDIT bedit

PERM filename perm filename

STATUS status

WAIT wait

DIR dir

dir arg

CLS cls

SYSTEM:"..." system:"@name@"

SYSTEMCURR:"..." systemcurr:"@name@"

SLEEP n sleep @name@

WAITBELL n waitbell @name@

DEBUG=ON debug=on

DEBUG=OFF debug=off

TRACE=ON trace=on

TRACE=OFF trace=off

$name $@name@

### Note

The Cursor Keys (Up/Down) will scroll through all interactive commands that have been entered.

## B-EDIT Function List

### -number

Desc : This is used when B-Edit is called to change the default maximum number of characters in a variable from 1000 to a different size. The number entered may be 100-20000.

Syntax : -number

Example : bedit -2500

bedit -2500 call example.bed

### !LABEL

Desc : The !Label is used to define program flow transfer points to B-Edit. You use labels in GOTO's or CALL's. B-Edit will begin at the top of the command file and search downward for the label called. You shouldn’t have two identical labels within the same command file, only the first label would be known to B-Edit.

See CALL, GOTO.

Syntax : Must begin in column 1 and begin with the '!' character.

No other command should be included on a label line.

Example : !A100\_Begin\_Processing

### $NAME

Desc : The $ indicates to B-Edit that the string variable name following the $ contains a dynamic command to execute (MACRO). You may place any B-Edit command in the $MACRO (CALL, WINDOW, etc...).

Syntax : $name

$@name@

Example : string testcmd

testcmd = /cursor 1,1/

$testcmd

### \*ENDDATA

Desc : This function is used to define the end of inline data. All lines following the BEDITDATA line (up to a \*ENDDATA line) will be inserted into the current file after the current line. Note : the \*ENDDATA must begin in column 1.

See BEDITDATA.

Syntax : \*enddata

Example : beditdata

...

\*enddata

### @

Desc : Used to work with the contents of a defined variable. Prefix the variable name with the @ and any function will refer to the contents of the variable. In most cases when referencing a variable using @name@ the @ following the name is optional. The case where the @ following the name is required is when using it in a string. i.e. print:"test1=[@test1@]" (This applies to PRINT, PRINTX,and variable assignment i.e. test = "@test1@ @test2@").If you leave off the second @ then the string will be unchanged i.e. test = "@test" + count, if count = 1 then test will be "@test1". This allows you to still use variables dynamically. You may want to have the last variable in the chain of @variables to point to a variable without changing the data to include the @. To do this final recursion pointing use @@variable. This works for assignment, but it will not work for printing since the @'s should enclose a variable name in the print. To print using this final recursion use @^ instead of @@. i.e.

Varptr1 = "Var2"

string @Varptr1 = "FINAL" // create string Var2 = 'FINAL'

string Var3 = @@Varptr1 // create Var3 = 'FINAL'

print:"@^Varptr1@" // print 'FINAL'

Syntax : @Varname

@@Varname

@^Varname

Example : print:"The contents of Var1 = [@Var1@]"

StrVar = "@Var1@-@Var2-@Var3@"

string @Varptr = "test"

string Var2 = @@Varptr

print:"@^Varptr@"

### ABS

Desc : Return the absolute value of a float variable.

Note: the return assignment variable must be a FLOAT variable.

See: ROUND, SQRT

Syntax : floatVariable = ABS (Var2)

Example : int Var2 = -10

float fAbsVar2 = abs(Var2)

### ACOS

Desc : Return the Arc Cosine of a number.

Note: the return assignment variable must be a FLOAT variable.

See COS, SIN, TAN, ASIN, ATAN

Syntax : floatAngle = ACOS (Val)

Example : float Value= .70

float fAngle = acos(Value)

### ARG

Desc : Used to create LOCAL variables in a subroutine that will receive passed arguments from a CALL. These variables will be erased after the RETURN just as LOCAL variables are erased. ARGuments should be created in the order they arepassed in the CALL. i.e. If the CALL passes a STRING thenan INTEGER, the subroutine should create an ARG STRING then an ARG INTEGER. You may create LOCAL variables before or after ARGs.

See CALL, LOCAL.

Syntax : arg string Varname

arg integer Varname

arg float Varname

arg dollar Varname

Example : arg string Argument1

### ASIN

Desc : Return the Arc Sine of a number.

Note: the return assignment variable must be a FLOAT variable.

See COS, SIN, TAN, ACOS, ATAN

Syntax : floatAngle = ASIN (Val)

Example : float Value = .70

float fAngle = asin(Value)

### ATAN

Desc : Return the Arc Tangent of a number.

Note: the return assignment variable must be a FLOAT variable.

See COS, SIN, TAN, ACOS, ASIN

Syntax : floatAngle = ATAN (Val)

Example : float Value = 1

float fAngle = atan(Value)

### B

Desc : Move backward in the current file.

Syntax : B

B;n

B;\*

B;@name@

Example : b

b;10

b;\*

integer count

count = 5

b;@count@

### BACK

Desc : This allows you to enter a string to back end of lines in the current file.

Syntax : BACK:/.../;#

back:/@var/;@var

back:/.../

### BEDIT

Desc : Enter the B-Edit system.

Syntax : BEDIT

If you are entering B-Edit from the PC command prompt you may enter a call of a filestring to execute that B-Edit program.

Example : BEDIT

BEDIT call c:\bedit\test1.bed

### BEDITDATA

Desc : This function is used to define the start of inline data. All lines following this line (up to a \*ENDDATA line) will be inserted into the current file after the current line. Note : the \*ENDDATA must begin in column 1.

See \*ENDDATA.

Syntax : beditdata

Example : beditdata

...

\*enddata

### BEGINEND

Desc : This function is used in the FS and PS commands. It allows you to search lines that BEGIN or END with the string you want to search for. This function may not be used in conjunction with the COLS command. Caution : Don't forget to turn this function off when you are finished with it.

Syntax : beginend begin

beginend end

beginend off

Example : beginend begin

### BELL

Desc : Ring the Bell

Syntax : BELL

Example : BELL

### BOOKMARK

Desc : This function is used to mark a spot in a file to allow easy access back to that line. You must create a bookmark string variable and assign the text you want to use as a unique bookmark. You may use this bookmark on as many lines as you want. You may assign as many different bookmarks to a file as you want. You may only assign one bookmark to one line in the file at any given point. To assign a bookmark to a line create the bookmark variable and assign the bookmark text to it, use BOOKMARK Varname (where Varname is the string variable you are using as the bookmark). The current line has now been bookmarked. When you want to access the line based on the bookmark use the string commands FS,PS,CUT, and COPY as you normally would but use the bookmark variable as the string to find (i.e. fs:/@Bookmark/). The FS,PS,CUT, and COPY functions work as normal, but since the variable searched for has been set up as a bookmark, only the bookmark will be searched rather than the actual text for each line. To remove a bookmark from the current line use BOOKMARK ERASE. To remove all bookmarks from the current file use BOOKMARK ERASEALL.

See FS,PS,CUT,COPY

Syntax : bookmark BookmarkVariable

bookmark erase

bookmark eraseall

Example : string Place1 = "ABC"

...

b ps:/Place Bookmark Here/

bookmark Place1

...

b

fs:/@Place1/

print:"We should now be at the bookmark line"

### BREAK

Desc : Turn the CTRL-BREAK key ON/OFF. If BREAK is turned ON when the CTRL-BREAK key is pressed program flow will be sent to the !BREAK label if it exists. Do not turn BREAK back ON in the !BREAK processing. Always finish the !BREAK processing with an EXIT. You may use any BEDIT function in the !BREAK processing. Processing will terminate immediately if you do not code the !BREAK. This is very useful if you want to clean up processing when a user breaks out of a process. You may want to turn BREAK OFF during critical processing.

Syntax : break=off

break=on

!BREAK

Example : break=off

critical processing...

break=on

normal processing...

!BREAK

break processing...

### BUFFER

Desc : Display the current contents of the cut/copy buffer.

See CUT, COPY, PASTE, CLEAR.

Syntax : buffer

Example : buffer

### CALL

Desc : Call a subroutine within the current B-Edit command file, or call a different B-Edit command file. When you launch B-Edit from the command line you can include the following parameters:

-MaxLength Instructs B-Edit to create variables with this length.

-E:Key Instructs B-Edit to decrypt the script file called using the key entered. Use –E:DEFAULT to use the default Encryption Key.

CALL filename Instructs B-Edit to read and execute the filename entered.

e.g. bedit call script1.bed

bedit -5000 call script1.bed

bedit –E:TestKey call encryptedfile.bed

The current command execution is preserved on a stack and will be restored when a matching return command is executed. If you use the suffix .BED on your B-Edit command files the .BED suffix is optional on the call. If the file CALLed is not in the current directory the BEDIT and then the DPATH set in the CONFIG.SYS file is used to locate the file. i.e. put "SET BEDIT=c:\bedit;w:\applic\bedit;" in the CONFIG.SYS. If you specify \* as the filename then the current file will be CALLed. You may call a label within a file by using CALL FILE!LABEL. You may pass up to 30 arguments into the called subroutine. After the call label enclose the arguments within brackets [] (the arguments may be entered on separate lines). In the subroutine create corresponding ARG variables. ARG variables will be erased after the RETURN just as LOCAL variables are erased. When the subroutine is called the arguments passed will be assigned to the ARG variables (in order).

i.e. call !a100\_sub1 [@String1 @Integer1 TEST]

...

!a100\_sub1

arg string Str1 // will equal what’s in String1

arg integer Int1 // will equal what’s in Integer1

arg string Str2 // will equal what’s in String2

local string Str3 // Local Variable

...

return

Note that in the example the first two arguments passed are the contents of variables, and the third argument is actual text. Arguments are delimited by space, so if you want to pass text into a subroutine that has spaces in it, assign the text to a variable and pass the contents of the variable. You may pass any of the variable types into the subroutine (STRING, INTEGER, FLOAT, DOLLAR). Make sure that you keep the variable types in order. i.e. If the first argument is a STRING, and the second argument is an INTEGER, then in the subroutine CALLed create an ARG STRING first, and an ARG INTEGER second. You may also assign a value to a variable returned by the subroutine.

i.e. RetStr = call !a100\_sub1 [TESTING123]

if (RetStr = "BAD") ...

...

!a100\_sub1

arg string Arg1

arg integer Arg2

local string ReturnVal

...

ReturnVal = "OK"

if (...) ReturnVal = "BAD"

return @ReturnVal

In this example the string RetStr is assigned what the subroutine returns (the returned string is evaluated in the following if statement). The subroutine has a variable following the RETURN statement. The contents of this variable are actually returned into the CALL variable (RetStr). You may use any variable type to return (STRING, INTEGER, FLOAT, DOLLAR). Make sure that you RETURN the correct type to the CALLing variable. i.e. If the CALLing variable is an INTEGER then return an INTEGER value.

See RETURN, PERM, ARG.

Syntax : call !label

call filename

call @name@

call \*

call filename!label

call !label [Arg1 Arg2 Arg3]

Example : call !A100\_Begin

call c:\bedit\test.bed

call c:\bedit\test

string test1

test1 = "!A100\_Begin"

call @test1@

call filea!Label1

RetVal = call filea!Label1 [Arg1 Arg2]

### CASE

Desc : Turn Case Sensitifity ON or OFF. If CASE is followed by ON then all tests will be Case Sensitive (i.e. Test is different than TEST). If CASE is followed by OFF then all tests will be Case Insensitive (i.e. Test is the same as TEST). This is handy when trying to 'FS' a character string but you don't know where the capitals are. Caution : If the Case tests are set to insensitive ALL TESTING is Case insensitive. This means that all variables, labels, etc will not be tested case sensitive. i.e. If you have two variables named Test1 and TEST1 they will not be distinguished when using them. Just be careful about the way you assign names and things should be fine when turning Case Sensitivity off.

Syntax : case on

case off

Example : case on

### CHAIN

Desc : Show the entire chain structure for a variable (Pointer variable structure using @, usually used in interactive).

See @.

Syntax : chain name

Example : chain test1

### CHAR

Desc : Creates a string which is the same as the STRING command.

See STRING.

Syntax : char name

Example : char test = ‘y’

### CLEANKEYS

Desc : Clear out the Keyboard Buffer. This is a handy way to ensure the user has not pressed any keys before INPUTing a critical field.

See INPUT.

Syntax : cleankeys

Example : cleankeys

input criticalfield

### CLEAR

Desc : Clear out the contents of the cut/copy buffer.

See BUFFER, CUT, COPY.

Syntax : clear

Example : clear

### CLOSE

Desc : Remove an open file from memory.

See OPEN, READ, FILE, WRITE.

Syntax : close name

close @name@

Example : close filea

### CLS

Desc : Clear the screen.

Syntax : cls

Example : cls

### COLOR

Desc : This function allows you to change the color of the characters printed in the DOS display window.

Syntax : color n (where n = 0 – 15)

color @IntVariable (contains 0-15)

color @StringVariable@

color COLORNAME

String variables or COLORNAME can equal:

BLACK black

DARKBLUE darkblue

DARKGREEN darkgreen

DARKCYAN darkcyan

DARKRED darkred

DARKMAGENTA darkmagenta

BROWN brown

GRAY gray

BLUE blue

GREEN green

CYAN cyan

RED red

MAGENTA magenta

YELLOW yellow

WHITE white

INVERSE inverse

DEFAULT default

Example : color 1 print:”Color 1”

string CharCol = ‘red’

color CharCol print:”This is red text”

color white print:”This is white text”

### COLS

Desc : Masks off the lines in a file for the PS, FS, RS, and RVS commands. i.e. You may only want to find a string in certain positions in a line. Using the COLS command you will mask off every character except the characters within the range you specify. Use NOCO to turn off the COLS masking. This function may not be used in conjunction with the BEGINEND function. Caution : Don't forget to turn this function off when you are finished using it.

See PS, FS, RS, RVS, NOCO.

Syntax : cols start,end

cols @var@,@var@

Example : cols 3,5

cols @start,@end

### COPY

Desc : Copy lines from the current file into the cut/copy buffer. If a PASTE has been done since the last CUT/COPY then the buffer is cleared out before the copy. You may also use COPY similar to the FS command, where you specify what string to search for, and if found the line is COPYed. This function uses Pattern Matching. The Default Pattern is ?=Any Character, #=Number, &=Alpha, ~=Special, \*=To the End. If you use one of the Pattern Matching characters, the Pattern Matching rules take over. Use the PATTERN command to change the default if you want to use one of the default Pattern characters in this function.

See CLEAR, CUT, PASTE, BUFFER, BOOKMARK

Syntax : copy

copy;n

copy;\*

copy;@name@

copy:/xxx/

copy:/xxx/;n

copy:/xxx/-/yyy/

copy:/xxx/+/yyy/;n

Example : copy;3

### COS

Desc : Return the Cosine of an angle of zero to360 degrees.

Note: the return assignment variable must be a FLOAT variable.

See SIN, TAN, ACOS, ASIN, ATAN

Syntax : floatVariable = COS (Angle)

Example : int Angle = 45

float fCosine = cos(Angle)

### CRUSH

Desc : Remove all spaces from a string variable.

Syntax : crush name

crush @name@

Example : string test1

test1 = "this is a test"

crush test1

### CURRDIR

Desc : This function will move the current directory location into the string variable given. This is very handy when you want to display the current directory to the user.

Syntax : currdir strvar

Example : string Currentdir

currdir Currentdir

print:"The current directory is [@Currentdir@]"

### CURSOR

Desc : Move the cursor to a 1 based (home = 1,1) Row,Col location on the screen. SAVE will save the current cursor position, and RESTORE will restore the last SAVEd cursor position. You must set the cursor position (print, cursor #,#, etc...) before using the cursor save/ restore function.

Syntax : cursor #,#

cursor @xname,@yname

cursor save

cursor restore

cursor up,#

cursor up,@name

cursor down,#

cursor down,@name

cursor down,#

cursor down,@name

cursor right,#

cursor right,@name

cursor left,#

cursor left,@name

Example : cursor 10,10

cursor @Row,@Col

cursor save cursor 10,10 print:”Here” cursor restore

### CUT

Desc : Cut lines from the current file into the cut/copy buffer. The lines will be removed from the current file. If a PASTE has been done since the last CUT/COPY then the buffer is cleared out before the cut. You may also use CUT similar to the FS command, where you specify what string to search for, and if found the line is CUT. This function uses Pattern Matching. The Default Pattern is ?=Any Character, #=Number, &=Alpha, \*=To the End. If you use one of the Pattern Matching characters, the Pattern Matching rules take over. Use the PATTERN command to change the default if you want to use one of the default Pattern characters in this function.

See CLEAR, COPY, PASET, BUFFER, BOOKMARK.

Syntax : cut

cut;n

cut;\*

cut;@name@

cut:/xxx/

cut:/xxx/;n

cut:/xxx/-/yyy/

cut:/xxx/-/yyy/;n

cut:/xxx/+/yyy/

cut:/xxx/+/yyy/;n

Example : cut;3

### D

Desc : Delete lines from the current file.

Syntax : d

d;n

d;\*

d;@name@

Example : d;3

### DEBUG

Desc : Writes Debug Display's to BEDIT.BUG file while a B-Edit Command file is being processed. You may review BEDIT.BUG after the B-Edit run is complete to find problems in the code.

Syntax : debug=on

debug=off

Example : debug=on

### DECRYPT

Desc: This function will decrypt a variable using the key that’s been set using ENCRYPTKEY. To DECRYPT the string properly you must set the ENCRYPTKEY to the same value as when the variable was ENCRYPTed. If you don’t set the ENCRYPTKEY then a default key will be used.

See ENCRYPT, ENCRYPTKEY

Syntax: DECRYPT Variable

decrypt @Variable

Example: decrypt @EncryptedVariable

print:”@EncryptedVariable@” // prints decrypted string

### DECRYPTFILE

Desc: This function will decrypt the current file using the ENCRYPTKEY. To ENCRYPT a file use the ENCRYPTFILE function.

This is very useful if a data file contains sensitive information you don’t want visible to anyone who might have access to the file (e.g. if the file contains a password).

See ENCRYPTKEY, DECRYPTFILE

### DIR

Desc : Allows you to perform a Directory without entering system:"dir". This command sould only be used interactively. You may follow the DIR with only one argument (i.e. dir \*.c).

Syntax : dir

dir arg

Example : dir

dir \*.c

dir >temp.dir

### DIRECTION

Desc : Allows you to change the direction of the file pointer when using

FS, PS, COPY, etc... To change the direction of file movement to move upwards use

DIRECTION UP, and to return the movement back to normal use DIRECTION DOWN. This is handy if you want to search upwards in a file from the current position. Remember that all directional movement will be reversed when you use DIRECTION UP, so remember to code accordingly, and DON'T forget to change the DIRECTION back to DOWN when you are finished with this special processing.

Syntax : direction up

direction down

Example : direction up

### DO

Desc : Beginning of a DO-UNTIL loop. All statements appearing between the DO and the UNTIL will be processed until the UNTIL condition is true. The condition testing is exactly the same as the IF condition testing (e.g. use of AND/OR is permitted). Indenting within the DO-UNTIL is not required but it will make the code more readable. If you want to nest DO-UNTIL loops you must suffix the DO and the UNTIL with matching names. i.e. DO-LOOP1 and UNTIL-LOOP1

See UNTIL, IF

Syntax : do

...

until (condition)

doNAME

...

untilNAME

Example : do Count ++ print:"Count = @Count@" until (Count > 10)

Count = 0

do

Count ++

Count2 ++

Print:"Count = @Count@"

until (Count > 10) or (Count2 > 100)

Count = 0

do-loop1

Count ++

Count2 = 0

do-loop2

Count2 ++

print:"@Count1@ @Count2@"

until-loop2 (Count2 > 10)

until-loop1 (Count1 > 10)

### DOLLAR

Desc : Create a dollar variable. This variable will allow you to assign dollar values (i.e. 1.23, -34.53) and process dollar type math against the field (i.e. Dollar1 = Dollar1 + .02). Do not use any reserved word to name a variable (i.e. IF, STRING,STATUS, LINE, LIST, etc...). The maximum number of characters for a dollar name is 100.

See LOCAL, STRING, INTEGER, FLOAT, ERASE.

Syntax : dollar name

dollar @name@

dollar name = 1.3

dollar @name = 2.32

Example : dollar dlr

dollar testdollar = 5.99 + 3.21

### DUMP

Desc : Dump the contents of a variable.

See STRING, INTEGER, FLOAT, DOLLAR, LOCAL.

Syntax : dump name

dump \*

dump @name@

dump \*

Example : dump test1

### ELSE

Desc : The ELSE logic will begin processing if the previous IF condition fails. Only one ELSE will apply to the entire IF statement. i.e. If any IF fails the ELSE logic will be processed.

For more complex IF-ELSE statements use:

IF-name and ELSE-name, making sure to match the “name” on the ELSE to the “name” on the corresponding IF. When using this IF/ELSE syntax don’t add additional statements to the IF or ELSE lines.

See IF, ENDIF.

Syntax : else

Example : if (var1 = var2) print:"var1 = var2" else print:"var1 != var2"

if (var1 = var2)

+ print:"var1 = var2"

+ else

+ print:"var1 != var2"

+ endif

### ENCRYPT

Desc: This function will encrypt a variable using the key that’s been set using ENCRYPTKEY. To DECRYPT the string properly you must set the ENCRYPTKEY to the same value as when the variable was ENCRYPTed. If you don’t set the ENCRYPTKEY then a default key will be used. This is a useful function when you want to encrypt sensitive data to be stored anywhere that might be visible to non-authorized people.

See DECRYPT, ENCRYPTKEY

Syntax: ENCRYPT Variable

encrypt @Variable

Example: string test = “This is a test”

encrypt test

print:”@Test@” // prints encrypted string

### ENCRYPTFILE

Desc: This function will encrypt the current file using the ENCRYPTKEY. To DECRYPT the file use the DECRYPTFILE function. This is a useful function when you want to encrypt sensitive data to be stored anywhere that might be visible to non-authorized people.

Note: This function can be used to encrypt a B-Edit script file, save the file, and when executing the file you must use “BEDIT -E:EncryptKey call ScriptFilename” where EncryptKey is the key string used when the file was encrypted (use –E:DEFAULT to use the default Encryption Key). This is very useful if the script file contains sensitive information like a password that you don’t want anyone running the script to see, you can give them the key to run the script, but they won’t be able to view the contents of the script. You can use this function in interactive mode to open, encrypt, and save a B-Edit script file. Make sure to keep an original file as a backup.

**Warning**: If you encrypt a file (e.g. a .BED file to protect information in the code) using the default Encryption Key you won’t be able to decryptfile.

Example: open Filea c:\test.dat

encryptfile

See ENCRYPTKEY, DECRYPTFILE

### ENCRYPTKEY

Desc: This function allows you to set the encryption key used when ENCRYPT or DECRYPT is used on a string.

See DECRYPT, ENCRYPT, ENCRYPTFILE, DECRYPTFILE

Syntax: ENCRYPTKEY BEditKey

Encryptkey VariableName

Example: ENCRYPTKEY BEditKey //Assign the string “BEditKey” to the encryption key

string KeyVar = “This is a test key”

encryptkey @KeyVar //Assign the contents of KeyVar to the encryption key

### ENDIF

Desc : This command is documentation only. It will make continuations of IF's easier to read. No processing will take place for this command.

See IF, ELSE.

Syntax : endif

Example : if (var = "testing")

+ print:"testing"

+ wait

+ endif

### EOL

Desc : Clear the screen from the current cursor position to the end of the line.

Syntax : eol

Example : eol

### ERASE

Desc : Erase a variable(s) from memory. After an erase the variable may no longer be accessed. If you want to erase the interactive command scroll stack enter ERASE CMDS. If you want to erase only variables that have not been previously SAVEd use ERASE UNSAVED. This may be handy to create standard variables, save these variables, create some work variables for temporary use, and then erase these temporary variables without caring what their names are.

See LOCAL, STRING, INTEGER, FLOAT, DOLLAR.

Syntax : erase name

erase \*

erase @name@

erase cmds

erase unsaved

Example : erase test1

### ERROR

Desc : Processing will continue after any error condition if an "ERROR"command is found after the command. This may be useful when you are not sure of the environment (i.e. close an unopened file). If there is an error in the command prior to the ERROR command processing will continue after the ERROR command. If no error is found before the ERROR command, processing will continue to the next line. The ERROR command may follow any part of the line. So, if an error occurs at the start of the line, and there aremultiple commands on the line the first ERROR command found after the error will begin processing. Keep this in mind when coding ERROR conditions.

Syntax : error

Example : close Filea error print:"The file can't be closed"

### EXIT

Desc : Exit the current level of B-Edit.

See BEDIT.

Syntax : exit

Example : exit

### EXTRACT

Desc : Allows you to Extract data from a variable starting at a given location for a given length. The data extracted will be placed in the variable 'mid', the data to the left of 'mid' will be placed in 'left', and the data to the right of 'mid' will be placed in'right'.

Syntax : extract var,start,length

extract @var@,@start@,@length@

Example : extract line,10,3

### F

Desc : Move foreward in the current file.

Syntax : f

f;n

f;\*

f;@name@

Example : f;3

### FILE

Desc : Change the current file to a different open file. All pointers within all files are maintained.

See OPEN, CLOSE, WRITE.

Syntax : file name

file @name@

Example : file file1

### FILES

Desc : Print a list of all open files.

See OPEN, READ, CLOSE, FILE, WRITE.

Syntax : files

Example : files

### FILESIZE

Desc : Change a file to fixed length record size. Any record that has a smaller record size will be padded with spaces, and any record larger will be truncated. If you give a END of 'VAR' then the file will be changed to variable length records (all trailing spaces will be removed).

See OPEN, READ, CLOSE, FILE, FILES, FILESTRING, WRITE.

Syntax : filesize filename;start;end

filesize @filename;@start;@end

filesize \*;start;end

filesize \*;1;var

Example : filesize file1;1;80

filesize \*;5;10

filesize \*;1;var

### FILESTRING

Desc : This function allows you to access loc location of a OPENed file. i.e. If the file in the OPEN statement was not found in the current directory but it was found in the SET BEDIT path (in the config.sys) you may need access to where the file was found. The filestring will be moved into the string variable given, if the file is OPEN, or NULL if the file is not open.

See OPEN, READ, CLOSE, FILE, WRITE.

Syntax : FILESTRING file stringvar

Example : string Wherefile

open test test.dat

filestring test Wherefile

print:"@Wherefile@"

### FIXFILE

Desc : Some files (e.g. JS files, are note separated with carriage returns, so lines in a file might be too large for B-Edit to process. To “fix” these type of files use the FIXFILE command, which reads a “long line” file, and writes a new file with carriage returns which can be processed by B-Edit. Note: If the filenames need to include a directory path then you must place the directory path in a variable and use that variable name.

See OPEN, READ, CLOSE, FILE, WRITE.

Syntax : FIXFILE FromFilename ToFilename

Example : FIXFILE long.js long.js1

sting @FromFilename = ‘c:\javascript\long.js’

string ToFilename = ‘c:\javascript\longfix.js’

fixfile @FromFilename @ToFilename

### FLOAT

Desc : Create a float variable. This variable will allow you to assign float values (i.e. 1.234, -34.5353) and process float type math against the field (i.e. Float1 = Float1 + .0232). Do not use any reserved word to name a variable (i.e. IF, STRING, STATUS, LINE, LIST, etc...). The maximum number of characters for a float name is 100.

See LOCAL, STRING, INTEGER, DOLLAR, ERASE.

Syntax : float name

float @name@

float name = 1.332

float @name = 2.3233

Example : float testflt

float testfloat = 5.993 + 3.215

### FRONT

Desc : This allows you to insert a string to the front end of lines in the current file.

Syntax : FRONT:/.../;#

front:/@var/;@var

front:/.../

### FS

Desc : Find a character string in the current file. If the '+' is used in the syntax it means that the line must contain both values. If the '-' is used in the syntax it means that the line must contain at least one of the values. This function uses Pattern Matching. The Default Pattern is ?=Any Character, #=Number, &=Alpha, ~=Special, \*=To the End. If you use one of the Pattern Matching characters, the Pattern Matching rules take over. Use the PATTERN command to change the default if you want to use one of the default Pattern characters in this function.

See PS,CUT,COPY,BUFFER,BOOKMARK

Syntax : fs:/xxx/

fs:/xxx/;n

fs:/xxx/+/yyy/

fs:/xxx/+/yyy/;n

fs:/xxx/-/yyy/

fs:/xxx/-/yyy/;n

fs:/@name@/;@name@

fs:/@name@/+/@name@/;@name@

fs:/@name@/-/@name@/;@name@

Example : fs:/testing 123/

### GOTO

Desc : Transfer control to the label named.

See !Label.

Syntax : goto !Label

goto @name@

Example : goto !A100\_Test

### I

Desc : Insert a line into the current file following the current line. Insert lines into the current file lines entered from the keyboard until a null line is entered. Insert all lines after the current line.

Syntax : i:/xxx/

I:/@name@/

i

Example : I:/This is a new line/

i

### IB

Desc : Insert a line into the current file before the current line. Insert lines into the current file lines entered from the keyboard until a null line is entered. Insert all lines before the current line.

Syntax : ib:/xxx/

ib:/@name@/

ib

Example : ib:/This is a new line/

ib

### IF

Desc : Branch the program logic based on the condition tested. If the condition within the parens is true then the rest of the line is processed(up to ELSE). If the condition within the parens is false then control is passed to the next line the command file, or to the first ELSE found on the line, or continuation lines. You may continue an IF statement on the next line by placing a '+' in the first column of the line. The ELSE processing will begin processing if the IF condition fails. Keep in mind that there is only one ELSE for each IF statement. i.e. IF (var1 = var2) IF (var3 = var4) print:"OK" else print:"BAD" in this example if either IF fails the message BAD will be printed. Just be careful with the IF/ELSE logic and keep them simple. You may want to place the IF statements on separate lines to make the code more readable.

i.e. if (var1 = var2) and (var3 = var4)

+ print:"OK"

+ else

+ print:"BAD"

+ endif

ENDIF is used for visual purposes only and doesn't process logic. You may string together many IF's using OR or AND's. i.e. if (var1 = "a") or (var1 = 'b') and (var2 = 'c') ... The AND works the same as another IF statement. AND conditions may be placed on separate lines using the + continuation character, but the OR should be on the same line as an IF or AND.

i.e. if (a = b)

+ and (c = d) or (e = f)

+ print:"Testing"

+ endif

You may use [var-var] to test portions of a variable. i.e. if (var1[1-3] = "123") ... (You may use "," or "|" as a separator). see description for "name" for more information on how [] works.

For more complex IF-ELSE statements use:

IF-name and ELSE-name, making sure to match the “name” on the ELSE to the “name” on the corresponding IF. When using this IF/ELSE syntax don’t add additional statements to the IF or ELSE lines.

E.g.

if-a (a = 'a')

+ if-b (b = 'b')

+ print:"if-b (a='a')(b='b') OK all"

+ print:"ok all 2"

+ else-b

+ print:"if-a else-b(a='a')(b!='b') bad b"

+ print:"bad b line 2"

+ else-a

+ if-c (a = 'ok')

+ print:"else-a if-c (a!='a')(a='ok') a = 'ok'"

+ print:"a-ok 2"

+ else-c

+ print:"else-a else-c (a!='a')(a!='ok')bad a"

+ print:"bad a line 2"

+ endif

See ELSE, ENDIF.

Syntax : if (EOF) (eof) @name@ all cases

(NOTEOF) (noteof)

(!EOF) (!eof)

(PM) (!pm)

(EXISTS Filename) (exists @name@)

(NOTEXISTS Filename) (notexists @name@)

(!exists filename) (!exists @name@)

(FILEOPEN filename) (!fileopen @filename)

(KEY) (key)

(NOKEY) (nokey)

(!KEY) (!key)

(NUMERIC name) (!numeric name)

(DIGIT name) (!digit name)

(SPECIAL name) (!special name)

(ALPHA name) (!alpha name)

(MASK EntryVar MaskVar) (!mask EntryVar MaskVar)

(found) (!found)

(name) values 'Y', 'YES', 'GOOD', 'OK','TRUE', 'T', !=0, integer

(!name) values !('Y', 'YES', 'GOOD', 'OK', 'TRUE', 'T', !=0, integer)

(VAREXISTS name) (!varexists @name)

(name != "...") not equal

(name != n) " "

(name != name) " "

(name < "...") (name < n)

(name < name) (name <= "...")

(name <= n) (name <= name)

(name = "...") (name = n)

(name = name) (name > "...")

(name > n) (name > name)

(name >= "...") (name >= n)

(name >= name) (name[x-y] = name2[x-y])

(numeric name[x-y]) (!numeric name[x-y)

(name beginswith name2) excluding spaces and tabs

(name beginswith ‘…’)

(name endswith name2) excluding spaces and tabs

(name endswith ‘…’)

Example : if (eof) goto !a100\_end

if (exists test.dat) print:"test.dat exists"

if (notexists test.dat) print:"test.dat doesn't exist"

if (key) print:"a key has been pressed"

if (nokey) print:"no key has been pressed"

if (!mask Test\_Win1\_Entry1\_EFOBJ Entry1Mask)

+ message = "Invalid Entry"

+ window message error ok

+ return

+ endif

if (var1 = var2) call !a100-var1-equal-var2-subroutine

if (var1[1-2] = "eq") or (var[1-2] = "EQ") print:"ok"

if (var1[1,2] = "eq") or (var[1,2] = "EQ") print:"ok"

if (var1[1|2] = "eq") or (var[1|2] = "EQ") print:"ok"

if (var1 = var2) print:"var1 = var2" else print:"var1 != var2"

if (var1 = var2)

+ print:"var1 = var2"

+ else

+ print:"var1 != var2"

+ endif

If (line beginswith ‘abc’)

If (string1 endswith ‘test’)

### INITIALIZE

Desc : Initialize the contents of a variable.

See STRING, INTEGER, FLOAT, DOLLAR.

Syntax : initialize name

initialize \*

initialize @name@

Example : initialize test1

### INPUT

Desc : Input from the keyboard and place the input into the variable. You may include an optional number after the variable name indicating the maximum amount of characters to allow input. 'functionkey' will contain the following values depending on the function key pressed terminating the input:

F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12

INSERT, DELETE, HOME, END, PAGEUP, PAGEDOWN,

UP, DOWN, TAB, BACKTAB.

Syntax : input name

input @name@

input name,#

input @name,@number

Example : printx:"Input:" input test1

printx:"Input:" input Corp,3

### INPUTC

Desc : Input one character from the keyboard and place the input into the variable. This function is similar to INPUT, but characters won’t be printed, and as soon as a single character has been entered it will be placed in the variable and processing continues to the next command.

Syntax : inputc name

inputc @name@

Example : printx:"Input:" inputc CharacterInput

### INPUTP

Desc : Input from the keyboard and place the input into the variable and mask the characters that have been entered. This is useful when you want to input password information. You may include an optional number after the variable name indicating the maximum amount of characters to allow input. See INPUT.

Syntax : inputp Password

inputp @pw@

input pw,#

input @PW,@PW\_Length

Example : printx:"Password:" inputp Password

printx:"Password:" inputp Password,8

### INTEGER

### INT

Desc : Create an integer variable. Values may range from -32768 to 32767.Do not use any reserved word to name a variable (i.e. IF, STRING, STATUS, LINE, LIST, etc...). The maximum number of characters for a integer name is 100.

See LOCAL, STRING, FLOAT, DOLLAR, ERASE.

Syntax : integer name

int @name@

integer name = 1

int @name = 2

Example : integer count

Int testint = 5

### LIST

Desc : Allows you to easily List the contents of the current file. You may list one line, you may list from a starting line number to the end, you may list from a starting line number to another line number, you may list one page starting with a line number, you may list from a starting line number to the end of the file. If you use LIST NEXT the next page will list, and if you useLIST PREV the previous page will list.

See OPEN, READ, FILES, WRITE.

Syntax : list

list page

list lineno

list startline-

list startline-page

list startline-endline

list next

list prev

Example : list

list 10

list 10-20

list 10-

list page

list 10-page

list prev

### LEFTJUST

Desc : Left justify the contents of a STRING variable.

See STRING.

Syntax : leftjust name

leftjust @name@

Example : leftjust test1

### LEN

Desc : Find the length of a STRING variable.

See STRING.

Syntax : varname = len(varname)

@varname = len(@varname)

Example : Length = len(string1)

### LOCAL

Desc : Cause a variable to be local. If you create LOCAL variables in a CALLed routine they will be relased after the CALLed routine RETURNs. To make a variable local use the verb LOCAL before STRING, INTEGER, FLOAT, or DOLLAR. If a variable is not made LOCAL it will remain until you manually ERASE it.

See STRING, INTEGER, FLOAT, DOLLAR, ERASE, ARG.

Syntax : local string varname

local integer varname

local float varname

local dollar varname

Example : call !a100\_routine

...

!a100\_routine

local string LocalVar1

...

return

### LOGFILE

Desc : Change the output logfile from the default BEDIT.LOG to a file you specify. This should be done before any writing is done to the logfile (one of the 1st lines in your code). If logfile is set to null then no data will be written to the Log.

Syntax : logfile filestring

logfile null

Example : logfile NEWSYS.LOG

logfile null

### LOG

Desc : Return the Log of a value.

Note: the return assignment variable must be a FLOAT variable.

See: SIN, COS, TAN, ACOS, ASIN, ATAN

Syntax : floatLog = Log (Value)

Example : int Value = 10

float fLog = log(Value)

### LOWER

Desc : Change all characters of a STRING variable to lower case.

See STRING.

Syntax : lower name

Lower @name@

Example : lower test1

### MD

Desc : Make a new directory.

Syntax : md directory

md @dir@

Example : md newdir

### MIX

Desc : Change all characters of a STRING variable to Mixed case.

See STRING.

Syntax : mix name

mix @name@

Example : mix test1

### name (variable name)

Desc : Assign a value to a variable. You may use [x-y] to assign part of the variable i.e. var1 = "testing" and you assign var2 = "@var1[2-4]@" then var2 will become "est". If you use [end-var] then the assignment will be from var to the end of the string. The FROM and TO variables may be variable names, or numbers. You may use "-", ",", or "|" as a separator character. Integer arithmetic will support multiple operators. i.e. (int1 = int2 + int3 / int4 \* int5). B-Edit will process the arithmetic strictly from left to right. In the example above int2 is added to int3 and placed in int1, then int1 is divided by int4, then int1 is multiplied by int5.

Another way to use DOLLAR math is to create the variable as type DOLLAR. i.e.

dollar Field1

Field1 = 1.23 + 3.21

This is also true for FLOAT type fields

See STRING, INTEGER, FLOAT, DOLLAR.

Syntax : name = name + name (valid operators + - \* /)

name = name + name - name \* name / name

@name@ = @name@ + @name@

name = n

@name@ = n

name = LEN(name)

@name@ = len(@name@)

name = name + "..."

name = name + "@name1@ @name2@..."

name = "@name[x-y]@..."

name = "@name[x,y]@..."

name = "@name[x|y]@..."

name = "@name" + name

@name@ = @name@ + "..."

name = name + n

@name@ = @name@ + n

name = name + name (+ - \* /)

name ++ (add 1 to the variable)

name -- (subtract 1 from the variable)

name += nameb (add nameb to name)

name -= nameb (subtract nameb from name)

name \*= nameb (multiply name by nameb)

name /= nameb (divide name by nameb)

Example : count = 0

count = count + 1

test = "abc"

test = test + "def"

test = "test2=[@test2@] test3=[@test3@]"

test = "@test" + count

test = "@test2[1-3]@"

test = "@test2[pos1-pos2]@"

test = "@test2[end-3]@"

test ++

test –

test += test2

test -= test2

test \*= test2

test /= test2

Note: If you string multiple operators/operands they will be processed in left to

Right order, they won’t have the normal arithmetic precedence.

e.g. For INTEGER A = 2 + 6 / 2 you will get 4 since 2 + 6 = 8, then 8 / 2 = 4

### NOCO

Desc : Turn off the column masking set on the the COLS command.

See COLS.

Syntax : noco

Example : noco

### NOTE

Desc : Place comments in a command file. All code following the note will not be executed.

Syntax : note:"..."

//

\\ ...

\_ ...

Example : note:"this is a note line"

// this is a note line

\\ this is a note line

\_ this is a note line

### NOVE

Desc : Turn verification off. When a non printing command (i.e. fs:/xxx/) is executed the line will not be printed.

See VERI.

Syntax : nove

Example : nove

### OPEN

Desc : Open and read a file into memory. This file will become the current file with the pointer placed at the first line. If the file OPENed is not in the current directory the BEDIT and then the DPATH set in the CONFIG.SYS file is used to locate the file.

i.e. put "SET BEDIT=c:\bedit;w:\applic\bedit;" in the CONFIG.SYS.

If the file name is already open then the filename given will be appended to the end of the OPEN file and the pointer will be placed at the first line of the appended data. The maximum number of characters for a file name is 100.The maximum number of characters on a record read is 1000.

see READ, CLOSE, FILES, WRITE.

Syntax : open name filename

open @name@ @filename@

open name null

open @name@ NULL

Example : open file1 c:\bedit\bedit.dat

### P

Desc : Print the contents of the current file.

Syntax : p

p;n

p;\*

p;@name@

Example : p;3

### PACK

Desc : Remove all trailing spaces from the contents of a STRING variable.

See STRING.

Syntax : pack name

pack @name@

Example : pack test1

### PASTE

Desc : Insert the contents of the cut/copy buffer into the current file after the current line.

See COPY, CUT, BUFFER, PASTEB.

Syntax : paste

Example : paste

### PASTEB

Desc : Insert the contents of the cut/copy buffer into the current file before the current line.

See COPY, CUT, BUFFER, PASTE.

Syntax : pasteb

Example : pasteb

### PATTERN

Desc : The PATTERN command is used in two ways. First, the PATTERN command may be used to turn pattern matchin ON, or OFF. The default is OFF. Second, PATTERN is used to change the Pattern Matching characters used in FS,PS,RV,RVS,SCAN. The order in which the characters following the PATTERN command are as follows :

1st character = match on any character

2nd character = match on numbers only

3rd character = match on alpha characters only

4th character = match on any special character

5th character = match to the end

The Default Pattern Matching characters are :

? = Match on any character

# = Match any number

& = Match any alpha character

~ = Match on special character

\* = Match to the end or next char i.e. scan:/@test/:/very\*test/

Caution : Don't forget to turn this function off when you are finished using it.

See FS, PS, RV, RVS, SCAN.

Syntax : pattern on

pattern off

pattern xxxxx

Example : pattern on

pattern ^9X|>

### PERM

Desc : Make a CALLed file PERManent. When a file is CALLed the file is loaded into memory and interpretation of the commands in the file begins. When a RETURN for the CALL is located in the file the file is released from memory and control is passed back to the initiating CALL line. If PERM is used to make a file permanent the file is not released from memory when the RETURN is found. The next time the file is CALLed it will not be reloaded from disk since it is already in memory. You may want to make any file CALLed many times PERManent. This will improve the speed of your B-Edit routines. You must PERM the exact name as you would be calling it (include the .BED extension on the file in the PERM command if you will be using it in the CALL lines). Do not use the !label in the PERM command (i.e. if you use CALL filename!label use PERM filename).

See CALL, RETURN.

Syntax : perm filename

Example : perm startup.bed

### PRINT

Desc : Print to the screen a string or the contents of a variable followed by a carriage return line feed.

See PRINTX.

Syntax : print:"..."

print:"@name@"

print:"@name1@ @name2@..."

Example : print:"this is a test"

print:"@test1@"

print:"test1=[@test1@] test2=[@test2@]"

### PRINTX

Desc : Print to the screen a string or the contents of a variable. No carriage return line feed will be printed.

See PRINT.

Syntax : printx:"..."

printx:"@name@"

printx:"@name1@ @name2@..."

Example : print:"this is a test"

print:"@test1@"

printx:"test1=[@test1@] test2=[@test2@]"

### PS

Desc : Find &Print a caracter string in the current file. If the '+' is used in the syntax it means that the line must contain both values. If the '-' is used in the syntax it means that the line must contain at least one of the values. This function uses Pattern Matching. The Default Pattern is ?=Any Character, #=Number, &=Alpha, ~=Special, \*=To the End.If you use one of the Pattern Matching characters, the Pattern Matching rules take over. Use the PATTERN command to change the default if you want to use one of the default Pattern characters in this function.

See FS,CUT,COPY,BUFFER,BOOKMARK

Syntax : ps:/xxx/

ps:/xxx/;n

ps:/xxx/+/yyy/

ps:/xxx/+/yyy/;n

ps:/xxx/-/yyy/

ps:/xxx/-/yyy/;n

ps:/@name@/;@name@

ps:/@name@/+/@name@/;@name@

ps:/@name@/-/@name@/;@name@

Example : ps:/testing 123/;5

### R

Desc : Replace the current line with a string or the contents of a variable. A carriage return line feed will be added to the end of the replaced line.

Syntax : r:/.../

r:/@name@/

Example : r:/this is the new line/

### RANDOM

Desc : This function will generate a random number within a range and store it in the variable. A minimum and maximum value must be given as part of the call. The range’s minimum value must be less than or equal to the range’s maximum value.

Syntax : RANDOM RandomVar,5,10 // Will create a random value between 5 – 10

random RandomVar,@Min,@Max

Example : integer RandomVar

integer Min = 5

integer Max = 10

random RandomVar,@Min,@Max

### RD

Desc : Remove an existing empty directory.

Syntax : rd directory

rd @dir@

Example : rd emptydir

READ

Desc : Open and read a file into memory. This file will become the current file with the pointer placed at the first line. If the file OPENed is not in the current directory the BEDIT and then the DPATH set in the CONFIG.SYS file is used to locate the file. i.e. put "SET BEDIT=c:\bedit;w:\applic\bedit;" in the CONFIG.SYS. If the file name is already open then the filename given will be appended to the end of the OPEN file and the pointer will be placed at the first line of the appended data.

The maximum number of characters for a file name is 100.

The maximum number of characters on a record read is 1000.

see OPEN, CLOSE, FILES, WRITE.

Syntax : read name filename

read @name@ @filename@

read name NULL

read @name@ NULL

Example : read file1 c:\bedit\bedit.dat

### RESTORE

Desc : RESTORE defined variables from earlier SAVE. You may RESTORE system defined variables (i.e. mid), all user defined variables, all variables, or just one user defined variable (as long as the variable RESTORE'd has already been SAVE'd. This may be handy if you want to restore a variables value to a previous state.

see SAVE.

Syntax : restore sysvars

restore uservars

restore \*

restore name

Example : restore sysvars

restore uservars

restore \*

restore testvar

### RETURN

Desc : Return to the last CALL command and continue processing. Used to return processing back after a CALL. If the CALL used a variable assignment (RetVal = CALL !Label) then thereturn should be followed by an @Variable, or TEXT to be returned into the calling variable.

see CALL, PERM.

Syntax : return

return @Variable

return VALUE

Example : return

return @RetVal

return OK

return 0

### RIGHTJUST

Desc : Right justify the contents of a STRING variable.

See STRING.

Syntax : rightjust name

rightjust @name@

Example : rightjust test1

### ROUND

Desc : Return the rounded value of a float variable.

Note: the return assignment variable must be a FLOAT variable.

See: ABS, SQRT

Syntax : floatVariable = ROUND (Var2)

Example : float Var2 = 4.4445

float fRoundVar = round(Var2)

### RS

Desc : Change a string to a different string within the current file.

Syntax : rs:/xxx/:/yyy/

rs:/xxx/:/yyy/;n

rs:/xxx/:/yyy/;\*

rs:/@name@/:/@name@/

rs:/@name@/:/@name@/;@name@

Example : rs:/abc/:/def/;3

### RV

Desc : Replace the current line with a string or the contents of a variable. A carriage return line feed will be added to the end of the replaced line. Print all changes. This function uses Pattern Matching. The Default Pattern is

?=Any Character, #=Number, &=Alpha, ~=Special, \*=To the End.

If you use one of the Pattern Matching characters, the Pattern Matching rules take over. Use the PATTERN command to change the default if you want to use one of the default Pattern characters in this function.

Syntax : rv:/.../

rv:/@name@/

Example : rv:/this is the new line/

### RVS

Desc : Change a string to a different string within the current file and print all changes. This function uses Pattern Matching. The Default Pattern is

?=Any Character, #=Number, &=Alpha, ~=Special, \*=To the End.

If you use one of the Pattern Matching characters, the Pattern Matching rules take over. Use the PATTERN command to change the default if you want to use one of the default Pattern characters in this function.

Syntax : rvs:/xxx/:/yyy/

rvs:/xxx/:/yyy/;n

rvs:/xxx/:/yyy/;\*

rvs:/@name@/:/@name@/

rvs:/@name@/:/@name@/;@name@

Example : rvs:/abc/:/def/;3

### SAVE

Desc : SAVE defined variables for later RESTORE. You may SAVE system defined variables (i.e. mid), all user defined variables, all variables, or just one user defined variable. This may be handy if you want to restore a variables value to a previous state.

see RESTORE.

Syntax : save sysvars

save uservars

save \*

save name

Example : save sysvars

save uservars

save \*

save testvar

### SCAN

Desc : Scan a variable from the left to right searching for a string.

If the string is found then the following variables are updated :

left - contains the characters to the left of the string that was found

mid - contains the string that was found

right - contains the characters to the right of the string that was found

found - a capital 'Y' is moved to this variable I

If the string is not found then the following variables are updated :

found - a capital 'N' is moved to this variable

If the scan command uses the 'word' identifier then the scanning will search and find all characters delimited by a space. If the scaned string is not found the entire string will be moved into the variable 'left'.

Syntax : scan:/xxx/:/yyy/

scan:/xxx/:word

scan:/@name@/:/@name@/

Example : scan:/this is a test/:/is a/

scan:/this is a test/:word

scan:/@test1@/:word

### SCANR

Desc : SCANR functions the same as SCAN except that it searches from right to left

Syntax : scanr:/xxx/:/yyy/

scanr:/xxx/:word

scanr:/@name@/:/@name@/

Example : scanr:/this is a test/:/is a/

scanr:/this is a test/:word

scanr:/@test1@/:word

### SCRAMBLE

Dec: This function will search a file using a scrambled search string when SCRAMBLE has been turned on. Once SCRAMBLE has been turned on the search and print functions will work assuming that the text entered has been scrambled (fs, ps, rs).

Syntax: SCRAMBLE ON

scramble off

Example: open a bedit.txt

scramble on

ps:/bdeit/;\* // This will print all occurrences of bedit since that’s a match for // the scrambled letters “bdeit”

### SIN

Desc : Return the Sine of an angle of zero to360 degrees.

Note: the return assignment variable must be a FLOAT variable.

See COS, TAN, ACOS, ASIN, ATAN

Syntax : floatVariable = SIN (Angle)

Example : int Angle = 45

float fSine = sin(Angle)

### SIZE

Desc : Change the size of the contents of a STRING variable.

See STRING.

Syntax : size name,n

size @name@,@name@

Example : size test1,5

### SLEEP

Desc : Pause processing for n number of milliseconds.

Syntax : sleep n

sleep @name@

Example : sleep 500

integer count

count = 1000

sleep @count@

### SORT

Desc : Sort an open file. You must specify if the sort is (A)scending or (D)escending. You must specify the Sort Starting Column Number. The DOS sort is used to sort the file so the file to be sorted is limited to 64K.

See OPEN, FILE, FILES, WRITE.

Syntax : sort name,A,start

sort @name@,D,start

sort \*,A,start

Example : sort file1,a,1

sort @file1ptr@,a,@startpos@

sort \*,d,2

### SPLIT

Desc : Split a variable from the left n characters. Update the following variables after the split is performed :

left - contains n characters of the variable

right - contains the characters to the right of the nth character in the variable.

Syntax : split:/xxx/;n

split:/@name@/;@name@

Example : split:/this is a test/;5

split:/@test1@/;@count@

### SPLITR

Desc : Functions the same as SPLIT only the variable is split from right to left.

Syntax : splitr:/xxx/;n

splitr:/@name@/;@name@

Example : splitr:/this is a test/;5

splitr:/@test1@/;@count@

### SQRT

Desc : Return the Square Root of a variable.

Note: the return assignment variable must be a FLOAT variable.

See SIN,COS,TAN

Syntax : floatVariable = SQRT (Var2)

Example : int Var1 = 10

float Var2 = 20

float fResult1 = sqrt(Var1)

float fResult2 = sqrt(Var2)

### STATUS

Desc : Display the current status of B-Edit commands (i.e. Call Level, B-Edit Level, Number of Variables, Number of Files, COLS, SUBS, etc...)

Syntax : status

Example : status

### STRING

Desc : Create a string variable. Do not use any reserved word to name a variable (i.e. IF, STRING, STATUS, LINE, LIST, etc...).

The maximum number of characters for a string name is 100.

See LOCAL, INTEGER, FLOAT, DOLLAR, ERASE.

Syntax : string name

string @name@

string name = '...'

string @name = '...'

Example : string test1

string teststr = "This is a test string"

### SUBS

Desc : Change the Variable Substitution Character from the default '@'.

See @.

Syntax : subs char

Example : subs &

print:"&test"

### SYSTEM

Desc : Execute a DOS system command and wait until it’s finished.

Syntax : ystem:"..."

system:"@name@"

Example : system:"dir \*.c >temp"

string test

test = "dir \*.c >temp"

system:"@test@"

### SYSTEMCURR

Desc : Execute a DOS system command and place the results into the current file.

Syntax : systemcurr:"..."

systemcurr:"@name@"

Example : systemcurr:"dir \*.c"

string test

test = "dir \*.c"

systemcurr:"@test@"

### TAN

Desc : Return the Tangent of an angle of zero to360 degrees.

Note: the return assignment variable must be a FLOAT variable.

See COS, SIN, ACOS, ASIN, ATAN

Syntax : floatVariable = TAN (Angle)

Example : int Angle = 45

float fTangent = tan(Angle)

### TRACE

Desc : Displays each command line before it is executed and also each command as it's processed. After each command is displayed a wait message is displayed and processing is paused until a key is pressed. If TRACE=LOG is used processing is not paused, and only the line processed will be displayed to the log file.

Syntax : trace=on

trace=off

trace=log

Example : debug=on

### UPPER

Desc : Change all characters of a STRING variable to upper case.

See STRING.

Syntax : upper name

upper @name@

Example : upper test1

### UNTIL

Desc : End of a DO-UNTIL loop. All statements appearing between the DO and the UNTIL will be processed until the UNTIL condition is true. The condition testing is exactly the same as the IF condition testing (e.g. use of AND/OR is permitted). Indenting within the DO-UNTIL is not required but it will make the code more readable. If you want to nest DO-UNTIL loops you must suffix the DO and the UNTIL with matching names. i.e. DO-LOOP1 and UNTIL-LOOP1

See UNTIL, IF

Syntax : do

...

until (condition)

doNAME

...

untilNAME

Example : do Count ++ print:"Count = @Count@" until (Count > 10)

Count = 0

do

Count ++

Count2 ++

Print:"Count = @Count@"

until (Count > 10) or (Count2 > 100)

Count = 0

do-loop1

Count ++

Count2 = 0

do-loop2

Count2 ++

print:"@Count1@ @Count2@"

until-loop2 (Count2 > 10)

until-loop1 (Count1 > 10)

### VERI

Desc : Turn verification on. When a non-printing command (i.e. fs:/xxx/) is executed the line will be printed.

See NOVE.

Syntax : veri

Example : veri

### WAIT

Desc : Print a WAIT... message and wait until any key is pressed.

See WAITBELL.

Syntax : wait

Example : wait

### WAITBELL

Desc : Print a WAIT... message, Sound the bell and keep sounding the bell every n seconds until a key is pressed.

See WAIT.

Syntax : waitbell n

waitbell @name@

Example : waitbell 10

### Window

Desc : Write the contents of an open file to a filestring. After the write the pointer is placed at the top of the file. If the filename is specified as \* then the current file will be written. If the filestring is \* then the file will be written to the current filestring.

See OPEN, READ, CLOSE.

Syntax : write name filestring

write @name@ @filestring@

write \* filestring

Desc : Used to access all Presentation Manager functions. You must open the WindowFile file

(Describes all windows and objects), and the MenuFile file (Desctibes all main window menu selections), at the start of your application. You should also create string for each

object that you want to process actions for (use the same STRING

name as appears in the MenuFile or WindowFile) and assign the functions

to process when an action takes place in the object. After all

of this setup is complete a WINDOW START will bring up the main

window and remain in a loop here until PM processing is complete

(i.e. a WINDOW STOP, or EXIT, ...). If you want to process some

functions as the main window is created you should create the

STRING WindowInit and assign to it the functions you want to process.

If you want to process some functions when the PM processing is

finishing you should create the STRING WindowWrap and assign to it

the functions you want to process. If you want a to process a

function based on a time interval you should create the STRING

WindowTimer and assign to it the functions you want to process.

If you want to process a function when a function key is pressed

you should create a STRING named WindowF1 ... WindowF12 and assign

to it the functions you want to process.

If you want to process a function when the Delete key is pressed

you should create a STRING named WindowDelete and assign

to it the functions you want to process. Typically this will be

done in a list box when you want to allow the user to press the

Delete key to delete the selected line. You will want to use

the GETFOCUS function to find what list box had focus.

If you want to process a function when the Insert key is pressed

you should create a STRING named WindowInsert and assign

to it the functions you want to process. Typically this will be

done in a list box when you want to allow the user to press the

Insert key to insert a new line. You will want to use

the GETFOCUS function to find what list box had focus.

If you want to process a function when the Escape key is pressed

you should create a STRING named WindowEscape and assign

to it the functions you want to process. Typically this will be

used to handle CLOSEing windows when the user presses ESC. If you

don't code a WindowEscape action the window with focus will

automatically be closed when the user presses Escape. If you code

a WindowEscape action you must use the GETFOCUSWIN function to

find the window that has focus, and PUSH the appropriate Cancel

button on the window.

If you want to capture all Mouse Movement messages you must create

a string named WindowMouse and assign to it the functions you want

to process. i.e. WindowMouse = "call !a100\_WindowMouse"

If you want to capture Mouse button press messages you must create

a string named WindowDownx (where x = 1-3) and assign to it the

functions you want to process.

i.e. WindowDown2 = "call !a100\_WindowDown2"

If you want to capture Mouse button release messages you must

create a string named WindowDownx (where x = 1-3) and assign to it

the functions you want to process.

i.e. WindowUp2 = "call !a100\_WindowUp2"

Using these Mouse messages you could create drag and drop code by

checking when a button is pressed, WINDOW SIZE an object based on

mouse position, until the button is released.

Special Window Variables :

Variable Use

---------------------- ----------------------------------

WindowInit Action used when Initialization

WindowWrap Action used when Wrapup

WindowTimer Action used when timer interval

WindowInsert Action used when Insert pressed

WindowDelete Action used when Delete pressed

WindowEscape Action used when Escape pressed

WindowFn (n = 1-12) Action used when Function key

pressed

WindowMouse Action used when Mouse Moves

WindowDownx (x = 1-3) Action used when Mouse Buttons

1-3 Pressed

No Message Boxes should be used with WindowDown1!

WindowUpx (x = 1-3) Action used when Mouse Buttons 1-

3 Released

WindowUp Action used when the Up Arrow key is pressed

WindowDown Action used when the Down Arrow key is pressed

WindowRight Action used when the Right Arrow key is pressed

WindowLeft Action used when the Left Arrow key is pressed

MainWindow Used in setting Main Window Title

WindowBannerSize INTEGER, Specifies the main window

banner

character size (0 or not defining

the INTEGER variable will result

in

the banner not being shown).

WindowBannerColor STRING, Specifies the color of the

main window

banner characters (uses same color

names

as WINDOW COLOR). If this variable

is not

defined and set then the default

color is BLACK.

WindowBannerLine1 STRING, Specifies the 1st banner

line text.

If this variable is not defined

and set then no

text will print in the 1st banner

line.

WindowBannerLine2 STRING, Specifies the 2nd banner

line text.

If this variable is not defined

and set then no text will print in

the 2nd banner line.

WindowBannerLine3 STRING, Specifies the 3rd banner

line text.

If this variable is not defined

and set then no text will print

in the 3rd banner line.To create a

banner on the main window you must

create the

WindowBannerSize variable and assign it a value

greater than zero.

You must also create each banner

line and assign the characters

that you want to display on the

main window. This must be done

before the WINDOW START command is

issued.

i.e.

integer WindowBannerSize = 85

string WindowBannerColor = "blue"

string WindowBannerLine1 = "Forrest Bentley"

string WindowBannerLine2 = " B-Edit (PM)"

string WindowBannerLine3 = " Demonstration"

You may dynamically turn the banner off by

setting the WindowBannerSize to 0.

For most menu selections you will want to open a window and begin

processing within that window. To do this create the menu selection

string and assign the code to process when the item is selected

i.e. string Sys\_Main\_Selection1\_MSOBJ = "call !a100\_selection1"

action Sys\_Main\_Selection1\_MSOBJ

!a100\_selection1

window open Sys\_Win1\_WIN

window center Sys\_Win1\_WIN

return

Every action processed sould end with a return as shown above.

Keep in mind that these variables should be created before the

WINDOW START. When an action occurs within an object (i.e. a menu

selection selected, button pressed, etc...) that an ACTION has been

assigned to the code assigned to the object is executed.

When running in this PM mode you will not be able to use certain

BEDIT functions such as WAIT, INPUT, CHAR, COLOR, CURSOR, WAITBELL,

HELP, TRACE=ON, CLS, LIST.

Anything PRINTed will be sent to the LOG file. This may be used when

debugging a program. You may also want to use TRACE=LOG to debug a

program.

When pressing the ENTER key the default button will be pressed. The

default button is set to be the first button object defined in a

window. You may set the default to a different button using the

WINDOW DEFAULT function.

To keep your code clear you should use a naming convention on all

variables and objects. i.e.

Window Names = Sys\_Win\_WIN Sys = System Prefix

Win = Window Name

Object Names = Sys\_Win\_Name\_Type Sys = System Prefix

Win = Window Name

Name = Field Name

Type = PBOBJ for push buttons

EFOBJ for entry fields

CBOBJ for check boxes

RBOBJ for radio buttons

LBOBJ for list boxes

GBOBJ for group boxes

STTXT for static text

COOBJ for combo boxes

MLOBJ for multi line entry

You must open files (Note : for menu selection objects, push buttons,

radio buttons, and check box objects

the ~ character will underscore the

letter in the object and the action will

take place when the ALT-character or

CNTRL-character is pressed).

MenuFile (Case is sensitive!)

Desc : Defines all Menu Selections on the main window.

When the application starts the system will scan the

MenuFile and create objects/variables for the ITEMS

defined. You will place ACTION's on each ITEM allowing

full control of what should take place when the ITEM

is selected. (This file is normally suffixed with .MNU).

If you have a .RC file already you may

BEDIT CALL MENUFILE which will convert the RC file into

the MenuFile. When the WINDOW START function is exectued

the system will read through the MenuFile and create

string variables for each item object found.

Syntax : TITLE,title on the main window (Main Title)

MENU,~Menuname (Define Pulldown)

ITEM,Sys\_Main\_Name\_MSOBJ,~Itemname (Define Item)

---- (Separator)

Example : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* MENU \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TITLE,Test Main Window Title

MENU,~File

ITEM,Test\_Main\_Selection1\_MSOBJ,~Selection 1

------------------------------

ITEM,Test\_Main\_Exit\_MSOBJ,E~xit

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* MENU \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MENU,~Help

ITEM,Test\_Main\_About\_MSOBJ,~About

WindowFile (Case is sensitive!)

Desc : Defines all Windows and Objects within the Windows.

When the application starts the system will scan the

WindowFile and create string variables for the objects

defined. You will place ACTION's on each Object allowing

full control of what should take place when the Object

is selected. You may use the DLGBOX editor to create all

windows. After you have created all of your windows and saved

the file run BEDIT CALL WINFILE to convert the .DLG file into

a WindowFile (Normally suffixed with .WIN). The order the

objects appear in the file are important in that this

will be the tabbing order. Radio buttons will be grouped

together until a different object is found. Only one radio

button may be selected at one time in one group. Keep this

in mind if you want two radio button groups. It is good

practice to place a group box around grouped radio buttons.

The group box could be placed between the groups to keep

them separated. When tabbing to a radio button group only

the first button will be tabbed to. Once in the radio button

group you may use the cursor up and down keys to move through

the radio buttons within the group. User Defined Buttons should

be given the text MYCONTROL.

Syntax : Type,Objname,loc-x,loc-y,size-y,size-x,Text

where Type = WINDOW = window

PBOBJ1 push button

EFOBJL entry field left justified

EFOBJC entry field centered

EFOBJR entry field right justified

CBOBJA check box

RBOBJ1 radio button

LBOBJ1 list box single selection

LBOBJM list box multi selection

GBOBJ1 group box

STTXT1 static text

STTXTF frame

STTXTR rectangle

COOBJS simple combobox

COOBJD dropdown combobox

COOBJL dropdown list combobox

MLOBJ1 multi line entry field

Note : After you have created the WindowFile and the

MenuFile (Must be suffixed with .WIN and .MNU) you

may use BEDITPM CALL GEN to generate a BEDITPM program

with full navigation. This may be a good idea just to

give you an example program to review.

PaintFile: See WINDOW PaintFile

Error Processing : Typically when a user presses a button you will code

an action for that button. In the action logic you will

obtain the Window fields and check each field. If a field

contains data that you do not want you want to use the BELL

and MESSAGE functions to inform the user of the error, then

use the SETFOCUS function to set the focus to the field that

was found to be in error. You may want to check all of the

fields, use the COLOR function to change the color on all

fields found in error, display the error MESSAGE, and set

the focus to the first field in error. Remember to change

all colors back to default if you use this approch.

Long Processing : If you have Action logic that requires a long time to

process you may want to inform the user of what's happening.

There are a few ways to accomplish this. Use the WINDOW

TITLE function on any Window (Typically the MainWindow), to

display a message of what is currently happening. Use a number

of these WINDOW TITLE functions to keep the user informed.

You could also open your own Information window (you defined

in the DLGBOX editor, with one entry field DISABLED) and

WINDOW MODIFY the entry field at crittical points in your

processing. You must display a WINDOW MESSAGE to force this

new window visible before WINDOW MODIFYing the entry field.

An alternative to this approch is to create disabled buttons

on the window, and WINDOW PUSH a continuation button when

you want fields refreshed, and in the continuation button's

action logic, continue with the long process. This may become

tricky, but you will be able to force all refresh messages to

be processed when you required.

When the long process is complete CLOSE the Information window.

In all cases you should set the WINDOW POINTER to WAIT at the

beginning of the long Action, and set it back to ARROW when

the Action is complete.

Syntax : WINDOW parameters...

Example : window open System\_Window\_WIN

### WINDOW ACTION

Desc : This function is used to assign a process to an object when an action takes place in the object (i.e. when a button is pressed). Define the Object string and place the code to be

processed (when the action takes place) into the string before using the ACTION

function. You may dynamically change an object's ACTION any place in your logic (after

the string variable has been defined).

OBJECT Action processed when...

-------------- --------------------------

Entry Field looses focus

Push Button pressed

Check Box checked or unchecked

Radio Button pressed

List Box double click

Window ESC is pressed

Note : Radio Button ACTIONS should not generate MESSAGE's or OPEN

windows!

Do not use this function in conjunction with WINDOW SELECTACTION

See WINDOW SELECTACTION

Syntax : window action objname

Example : string Sys\_Win\_Ent1\_EFOBJ = "call !a100\_Sys\_Win\_Ent1\_EFOBJ\_focus"

window action Sys\_Win\_Ent1\_EFOBJ

### WINDOW AUTOTAB

Desc : This function is used to turn on/off autotabbing out of entry fields when they have been filled up. Some users may want to automatically tab to the next field when an entry field is completely filled in. This allows more rapid entry since the user doesn't have to press

the tab key to move to the next field. You may turn AUTOTAB on or off at any time

during execution. The size of an entry field is set when you use the WINDOW EDIT

or WINDOW MASK functions. See WINDOW EDIT, WINDOW MASK.

Syntax : window autotab on

window autotab off

Example : window autotab on

### WINDOW BACKCOLOR

Desc : This function is used to change any object's background color. This function should be called after the object's window has been OPENed. See WINDOW COLOR

Syntax : window backcolor objname color (where color = default, white, black,

background, blue, red, pink, green, cyan, yellow, neutral,

darkgrey, darkblue, darkred, darkpink, darkgreen, darkcyan,

brown, palegray, true, false)

Example : window backcolor Sys\_Win\_Entry\_EFOBJ red

### WINDOW CENTER

Desc : This function is used to center a window on the main window after it has been OPENed.

If this function is not used then the window that is OPENed will be located on the screen

exactly where it was created in the DLGBOX editor.

see WINDOW OPEN, WINDOW CENTERDESKTOP

Syntax : window center winname

Example : window center Sys\_Win\_WIN

### WINDOW CENTERDESKTOP

Desc : This function is used to center a window on the desktop after it has been OPENed.

If this function is not used then the window that is OPENed will be located on the screen

exactly where it was created in the DLGBOX editor.

see WINDOW OPEN, WINDOW CENTERDESKTOP

Syntax : window center winname

Example : window center Sys\_Win\_WIN

### WINDOW CHECK

Desc : This function is used to put a check mark next to a menu selection.

See WINDOW UNCHECK.

Syntax : window check msobj

Example : window check Sys\_Main\_Selection\_MSOBJ

### WINDOW CLEAR

Desc : This function is used to clear the variables associated with the objects on the screen and

also clear all objects on the screen. Combo boxes will not be cleared. You must clear

these manually since you probably don't want them cleared when the screen is cleared.

see WINDOW OPEN

Syntax : window clear winname

Example : window clear Sys\_Win\_WIN

### WINDOW CLOSE

Desc : This function is used to close a window after all processing is complete in the window.

i.e. when a Cancel button is pressed. Even though the window is closed all string

variables associated with the window objects are still available.

see WINDOW OPEN

Syntax : window close winname

Example : window close Sys\_Win\_WIN

### WINDOW CLOSEWINDOWNAME

Desc: This function is used to close a window on the desktop by the name that appears in the title of the window. All windows on the desktop will be enumerated and searched looking for a window name that matches the name in the variable provided, and once a match is found that window will be closed. This function is case sensitive, so you must enter the exact window name in the provided variable. You don’t need to match the entire window name, you can provide just the last portion of the name, e.g. if the window name is c:\users\213000864\BEDIT.exe, then you could just use “BEDIT.exe” and the correct window will be closed.

Syntax: window closewindowname closewinnamevar

Example: string CloseWinName = ‘BEDIT.exe’

Window closewindowname CloseWinName

### WINDOW COLOR

Desc : This function is used to change any object's foreground color. This function should be

called after the object's window has been OPENed.

Note : You can't COLOR a DISABLEd object.

See WINDOW BACKCOLOR

Syntax : window color objname color (where color = default, white, black,

background, blue, red, pink, green, cyan, yellow, neutral,

darkgrey, darkblue, darkred, darkpink, darkgreen, darkcyan,

brown, palegray, true, false)

Example : window color Sys\_Win\_Entry\_EFOBJ red

### WINDOW CONCEAL

Desc : This function is used to CONCEAL an entry field such as a password field. All characters entered will not be visible, but will still be accessable after an OBTAIN. You Must CONCEAL an entry field just before it's window is OPENed! Only entry fields that have

been CONCEALed before the OPEN will actually be CONCEALed.

see WINDOW UNCONCEAL.

Syntax : window conceal Sys\_Win\_Entry\_EFOBJ

Example : window conceal Sys\_Win\_Password\_EFOBJ

window open Sys\_Passwordwin\_WIN

### WINDOW CREATECONTROL

Desc : This function is used to create controls on a window (e.g. Push Buttons, Radio Buttons, etc…). Most windows and controls are created at startup based on the .WIN file definitions, but you can use CREATECONTROL to dynamically create a control as needed.

see WINDOW DELETECONTROL, WINDOW CREATEWINDOW.

Syntax : window createcontrol windowname controlname x y cx cy type text

Where windowname = window to create the control inside

controlname = name of the control to be created

x = horizontal position of the control

y = vertical position of the control

cx = width of the control

cy = height of the control

type = type of the control

PBOBJ1 push button

EFOBJL entry field left justified

EFOBJC entry field centered

EFOBJR entry field right justified

CBOBJA check box

RBOBJ1 radio button

LBOBJ1 list box single selection

LBOBJM list box multi selection

GBOBJ1 group box

STTXT1 static text

STTXTF frame

STTXTR rectangle

COOBJS simple combobox

COOBJD dropdown combobox

COOBJL dropdown list combobox

MLOBJ1 multi line entry field

Example : window createcontrol A\_WIN @A\_EFOBJ @X @Y @CX @CY @Type @Text

### WINDOW CREATEWINDOW

Desc : This function is used to create windows dynamically. Most windows and controls are created at startup based on the .WIN file definitions, but you can use CREATEWINDOW to dynamically create a window as needed.

see WINDOW DELETECONTROL, WINDOW DELETEWINDOW.

Syntax: window createwindow windowname x y cx cy title

Where windowname = window to create

x = horizontal position of the window

y = vertical position of the window

cx = width of the window

cy = height of the window

title = title for the window

Example : window createwindow A\_WIN @X @Y @CX @CY @Title

### WINDOW DELETE

Desc : This function is used to delete items from list boxes and combo boxes. You must assign to the string variable "ALL" if all lines in the object should be deleted, "INDEX=n" to delete item n from the list where n is the item number to delete (1-max).

If "ALL" or "INDEX=n" is not in the variable when the delete is processed then the hilited item in the object will be deleted.

see WINDOW INSERT, WINDOW SELECT

Syntax : window delete objname

Example : Sys\_Win\_List\_LBOBJ = "ALL"

window delete Sys\_Win\_List\_LBOBJ

Sys\_Win\_List\_LBOBJ = "INDEX=1"

window delete Sys\_Win\_List\_LBOBJ

### WINDOW DELETECONTROL

Desc : This function is used to delete a dynamically created control from a window.

see WINDOW CREATECONTROL, WINDOW CREATEWINDOW

Syntax : window deletecontrol @ControlName

Example : window deletecontrol Sys\_Win\_List\_LBOBJ

### WINDOW DELETEWINDOW

Desc : This function is used to delete a dynamically created window and all of the controls that exist on the window.

see WINDOW CREATEWINDOW, WINDOW CREATCONTROL

Syntax : window deletewindow @WindowName

Example : window deletecontrol Sys\_Windowname\_WIN

### WINDOW DEFAULT

Desc : This function is used to change the default push button (button used if you press Enter). The default ring will switch from the current default button (initially this is the first button in the tabbing sequence for a window) to the desired button.

Syntax : window default buttonname

Example : window default Sys\_Win\_Button1\_PBOBJ

### WINDOW DISABLE

Desc : This function is used to grey out and inactivate any object on a window. This is good practice when you don't want any action to take place for an object when it is not being used. This is also used to protect entry fields that you want to show but not change.

Objects that have been disabled will not be tabbed into. This function will work on any object including Menu Selections.

see WINDOW ENABLE

Syntax : window diable objname

Example : window disable Sys\_Win\_Entry1\_EFOBJ

### WINDOW EDIT

Desc : This is used to set up the editing that should be done in entry

fields (i.e. numeric only) and the format the field should take

when tabbing out (i.e. Mixed case, date mm-dd-ccyy). Define the

variable STRING and then set the EDIT. This should be done before

the WINDOW START. You will also define the size of the field to

restrict the number of characters allowed in the entry field.

You may dynamically set an object's EDIT any place in your logic

(after the string variable has been defined).

Note : If you set a WINDOW EDIT of DOLLAR any assignment value

should not use delimiters (i.e. DO NOT USE : Val = '1.23'

USE : Val = 1.23)

See Name (Assignment), WINDOW MASK

Syntax : window edit objname TYPE size

where TYPE = NUMERIC numeric chars only

= ZERO same as NUMERIC but

left zero fill to size of field

ALPHA alpha characters only

UPPER abc -> ABC

LOWER ABc -> abc

MIX john doe -> John Doe

DATE 1.2 -> 01-02-ccyy (curr year)

1.2.92 -> 01-02-1992

CURRDATE same as DATE but

blank -> mm-dd-ccyy (current)

DOLLAR 1234.5 -> 1234.50

-123 -> -123.00

TELEPHONE 7248123 -> 724-8123

724-8123 -> 724-8123

6127248123 -> (612) 724-8123

(612)7248123 -> (612) 724-8123

(612)724-8123 -> (612) 724-8123

(612) 7248123 -> (612) 724-8123

(612) 724-8123 -> (612) 724-8123

Note : If you define a field as DOLLAR you will be able to process

math (+, -, \*, /) with two decimal accuracy.

i.e. 1.23 + 3.21 = 4.44

Example : window edit Sys\_Win\_Field\_EFOBJ NUMERIC 5

### WINDOW ENABLE

Desc : This function is used to reENABLE an object after it has been DISABLED.

See WINDOW DISABLE

Syntax : window enable objname

### WINDOW EXISTS

Desc : This function is used to check to see if a window or control exists. This is useful to check to verify if the window or control exists prior to processing a function against the window or control. After the WINDOW EXISTS command has processed the result is placed in a variable named “status” which will contain ‘y’ if the window or control exists, and ‘n’ if the window or control doesn’t exist.

See: WINDOW CREATEWINDOW, WINDOW CREATECONTROL

Syntax : window exists Test\_Window\_WIN

Window exists Test\_Window\_Name\_EFOBJ

Example : window exists Test\_Window\_Name\_EFOBJ

If (status) window deletecontrol Test\_Window\_Name\_EFOBJ

### WINDOW FILEINSERT

Desc : This function is used to insert a file into list boxes and combo

boxes. You may insert lines at the END of the list or sort the line

into the object ASCENDing or DESCENDing. Make the file you want to

insert current (i.e. file InsertFile) before using this function.

You may want to DISABLE the object, INSERT the file, then ENABLE

the list. This will speed up the process since PM will not paint each

line as it is inserted into the list. You must also specify the color

of the line to be inserted (actual color values only apply to listbox's

when WINDOW LISTFONT has been set). You may set the color to any of

the following values : default, white, black, background, blue, red,

pink, green, cyan, yellow, neutral, darkgray, darkblue, darkred,

darkpink, darkgreen, darkcyan, brown, palegray, true, false.

see WINDOW DELETE, WINDOW SELECT

Syntax : window fileinsert end color objname

window fileinsert ascend color objname

window fileinsert descend color objname

Example : file InsertFile

window fileinsert end default Sys\_Win\_List\_LBOBJ

close InsertFile error

open InsertFile null

beditdata

Data1

Data2

Data3

\*enddata

window fileinsert ascend red Sys\_Win\_List\_LBOBJ

close InsertFile

### WINDOW FOCUSWINDOWNAME

Desc: This function is used to set focus to a window on the desktop by the name that appears in the title of the window. All windows on the desktop will be enumerated and searched looking for a window name that matches the name in the variable provided, and once a match is found, focus will be set to that window. This function is case sensitive, so you must enter the exact window name in the provided variable. You don’t need to match the entire window name, you can provide just the last portion of the name, e.g. if the window name is c:\users\213000864\BEDIT.exe, then you could just use “BEDIT.exe” and the correct window will gain focus.

Syntax: window focuswindowname focuswinnamevar

Example: string FocusWinName = ‘BEDIT.exe’

Window focuswindowname FocusWinName

### WINDOW FONT

Desc : This function is used to change the Font used for any Object on

a Window The window MUST BE OPEN before calling this function for

a non listbox object. If you want to set a global LISTFONT, but

still want to set one or more special listboxes with a different

FONT you must use the WINDOW FONT for the special listboxes before

OPENing the window that contains them (normally done in the

WindowInit logic), and the use the WINDOW FONT again for the

special listbox after the window is OPEN.

Use WINDOW LISTFONT only when you want to change the color of a

specific line in a listbox. You may use any of the following

fonts (Case sensitive) : 8.Courier

10.Courier

12.Courier

8.Helv

10.Helv

12.Helv

14.Helv

18.Helv

24.Helv

10.Fixed

12.Fixed

12.Proportional

8.Times

10.Times

12.Times

14.Times

18.Times

24.Times

See WINDOW LISTFONT.

Syntax : window font Objectname font

Example : window font Sys\_Win\_Entry\_EFOBJ 10.Times

window font Sys\_Win\_MLE\_MLOBJ 12.Courier

window font Sys\_Win\_List\_LBOBJ 10.Fixed

### WINDOW GETFOCUS

Desc : This function is used to retrieve the name of the object that has the

current focus. Typically this is used in the function key processing

to locate what field had the focus when the function key was pressed.

The name of the object that has the focus will be placed into the

variable name following the GETFOCUS.

set WINDOW GETFOCUSWIN

Syntax : window getfocus varname

Example : !WindowF1\_Processing

window getfocus FocusField

if (FocusField = "Sys\_Win\_Entry1\_EFOBJ")

### WINDOW GETFOCUSWIN

Desc : This function is used to retrieve the name of the window that has the

current focus. Typically this is used in the ESCAPE key processing

to locate what window had the focus when the ESCAPE key was pressed.

The name of the window that has the focus will be placed into the

variable name following the GETFOCUS.

Syntax : window getfocuswin varname

Example : !WindowEscape\_Processing

window getfocuswin FocusField

if (FocusField = "Sys\_Win\_WIN") window push Sys\_Win\_Cancel\_PBOBJ

### WINDOW HIDE

Desc : This function is used similar to DISABLE only the object is also made

invisible. If this function is used on a Menu Selection the Menu Selection

is destroyed and is not recoverable using a WINDOW SHOW.

If you use “window hide MAIN” the main window will be hidden. Caution should be used when hiding the main window since there may not be a way to shut the program off gracefully, in which case you’ll have to use the Task Manager to shut down the BEDITWIN.EXE process.

see WINDOW SHOW

Syntax : window hide objname

Example : window hide Sys\_Win\_Entry1\_EFOBJ

### WINDOW HWND

Desc : This function is used to convert an Object's HWND into a Variable

string for use in a USER defined routine.

Syntax : window hwnd Sys\_Win\_Field\_Type StringVariable

Example : window hwnd Sys\_Win\_Entry\_EFOBJ EntryHWND

### WINDOW INSERT

Desc : This function is used to insert lines into list boxes and combo

boxes. You may insert lines at the END of the list or sort the line

into the object ASCENDing or DESCENDing. Place the data that you want

to insert, into the variable name, and the process the INSERT function.

You may want to DISABLE the object, INSERT all the lines, then ENABLE

the list. This will speed up the process since PM will not paint each

line as it is inserted into the list. You must also specify the color

of the line to be inserted (actual color values only apply to listbox's

when WINDOW LISTFONT has been set). You may set the color to any of

the following values : default, white, black, background, blue, red,

pink, green, cyan, yellow, neutral, darkgray, darkblue, darkred,

darkpink, darkgreen, darkcyan, brown, palegray, true, false.

see WINDOW DELETE, WINDOW SELECT

Syntax : window insert end color objname

window insert ascend color objname

window insert descend color objname

Example : Sys\_Win\_List\_LBOBJ = "This is an inserted line"

window insert end default Sys\_Win\_List\_LBOBJ

window insert ascend red Sys\_Win\_List\_LBOBJ

### WINDOW LISTFONT

Desc : This function is used to change the Font used for all Listbox

Objects in the application. You must use this function if you

want to set different lines in a listbox to different colors. You

may also want to use this function when you want to use a different

fixed font than is supplied with the WINDOW FONT funtcion. All

fonts used in LISTFONT are fixed rather than proportional. You must

set the LISTFONT before calling WINDOW START.

DO NOT use this function in conjunction with WINDOW FONT on a listbox.

You may set the LISTFONT to a Fixed/Proportional Font numbered

1 through 8 (You'll have to try each to see which you like).

Each font will give a different sized font to work with.

Only call this function once.

See WINDOW FONT.

Syntax : window listfont font

Example : window listfont 1

### WINDOW MASK

Desc : This function is used to mask an entry field to allow only certain

characters to be enterered in certain positions. i.e. You may want

to mask a field to be 3 numerics and then 3 alpha characters. This

function uses the PATTERN function characters for pattern matching,

or exact characters. You can set up a MASK on an entry field

anytime after the WINDOW START. In most cases you will create

a STRING variable to contain the MASK, assign the MASK to the

entry field, and when a critical process takes place (i.e. OK Push

button) you will use the IF (!MASK EntryField MaskField) logic

to check the field before processing. The MASK can be any

character. If the MASK character is a PATTERN character then the

character entered in that position must match the PATTERN. If the

MASK character is not a PATTERN character then the character

entered in that position must match exactly (case sensitive).

If any character violates the MASK a warning will sound,

and if the character is the last character in the entry field

it will be erased.

The default PATTERN characters are :

? = Match on any character

# = Match any numeric

& = Match any alpha character

~ = Match on special character

Note : The 'Match to End' character doesn't apply to this function.

See PATTERN, WINDOW EDIT.

Syntax : window mask EntryFieldVar MaskVar

Example : window mask Test\_Win1\_Entry1\_EFOBJ Entry1Mask

### WINDOW MESSAGE

Desc : This function is used to display a message box. You have a wide selection

of icons to display in the message box as well as a number of buttons

to place within the message box. Place the message you want to display

into the system defined variable 'message', process the MESSAGE function,

and then test the 'message' variable for the button name that was pressed.

The name of the button pressed (Upper case) will be placed into the

variable 'message' after a button is pressed in the message box.

Syntax : window message icon buttons where icon = info (!)

warning (!), and beep

error (stop), and beep

question (?)

buttons = ok

okcancel

enter

entercancel

retrycancel

yesno

yesnocancel

abortretryignore

Example : message = "Do you want to exit?"

window message warning yesno

if (message = 'YES') exit

note : You could use 'if (message) exit'

since 'YES' in message will be true.

### WINDOW MODIFY

Desc : This function is used to move data from the string variables onto

the screen. Move the data into the string(s) and process this function

and the data will be displayed onto the screen. If the object is a

window then all objects on the window will be updated as if a MODIFY

was processed on each object.

Check Boxes and Radio buttons are handled specially. If the variable

contains a 'Y' the object will be turned on (checked), and if the

variable contains an 'N' the object will be turned off. If the

variable doesn't contain a 'Y' or an 'N' then the actual text of

the object will be changed to the string in the variable (include

the '~' character if you want to use accelerators on the object).

Push Buttons will not be modified when modifying the entire window.

If you want to change the text for a Push button you must modify

the object all by itself. Keep in mind that Push Button's have an

action associated with then so you must restore that action after

you have modified the text. i.e.

HoldButtonAction = Test\_Win1\_Button\_PBOBJ

Test\_Win1\_Button\_PBOBJ = "~New Text"

window modify Test\_Win1\_Button\_PBOBJ

Test\_Win1\_Button\_PBOBJ = HoldButtonAction

In this example the action associated with the Push Button is

assigned to the string HoldButtonAction, and after modifying the

buttons text, the action is restored. Note the '~' is included

in the new text for the button to allow Alt/Cntrl processing.

This process should be followed for Menu Selections when changing

Menu Selection text.

see WINDOW OPEN

Syntax : window modify objname

window modify winname

Example : Sys\_Win\_Entry\_EFOBJ = "New Data"

window modify Sys\_Win\_Entry\_EFOBJ

### WINDOW MOUSEPOS

Desc : This function is used to retrieve the current mouse coordinates.

See WindowDown1, WindowDown2, WindowDown3, WindowUp1, WindowUp2,

WindowUp3, WindowMouse

Syntax : window mousepos Xvar Yvar

Example : window mousepos XMousePos YMousePos

### WINDOW OBTAIN

Desc : This function is used to retrieve data from the screen into the

associated string variables. If the object is a window then every

object's screen data will be move into each associated string variable.

Check Boxes and Radio Buttons are handled specially. When you

obtain these objects the string will contain a 'Y' if the object

is selected (checked), and 'N' if it is not selected.

List boxes are handled specially. Move "COUNT" into the list box

string variable before the OBTAIN and the total number of items in the list

object will be returned into the string variable. Move "INDEX=n" into

the list box string variable before the OBTAIN and line n's

line will be returned into the string variable (when n = 1-max).

Move "INDEX" into the list box string variable before the OBTAIN and

the Index of the Hilited line in the list will be returned, 0 if no

line is selected.

For multi select list boxes move "FIRST" into the string before OBTAIN

and the first hilited line will be returned. Move "NEXT" into the

string before the next OBTAIN and the next hilited line will be

returned. i.e. Sys\_Win\_List\_LBOBJ = "FIRST"

window obtain Sys\_Win\_List\_LBOBJ

if (Sys\_Win\_List\_LBOBJ = null) goto !a100\_done

print:"@Sys\_Win\_List\_LBOBJ@"

!a100\_loop

Sys\_Win\_List\_LBOBJ = "NEXT")

window obtain Sys\_Win\_List\_LBOBJ

if (Sys\_Win\_List\_LBOBJ = null) goto !a100\_done

print:"@Sys\_Win\_List\_LBOBJ@"

goto !a100\_loop

!a100\_done

If you are deleting items in a multi select list you should delete

the line, and then use obtain "FIRST", until finished.

Syntax : window obtain objname

Example : window obtain Sys\_Win\_Entry\_EFOBJ

### WINDOW OPEN

Desc : This function is used to open a window and present the fields on

the screen. You may open a window with a SHADE pulled so the window

is not visible until you UNSHADE the window. Opening a window

with the SHADE pulled is the typical way to open a window. You want

the SHADE pulled until the window is CENTERed and all data has been

placed into the fields. After all of the data is in the window

you will UNSHADE the window to present it properly. Remember to

set the window pointer to WAIT while you load the window fields,

and restore the pointer back to ARROW after you UNSHADE. If you do

not use the WINDOW CENTER function, the window will be located on

the screen exactly where it was created using the DLGBOX editor.

All fields on the window will be enabled when the window is OPENed.

see WINDOW CLOSE, WINDOW CENTER

Syntax : window open winname shade

Example : window open Sys\_Win\_WIN shade

### WINDOW PAINT

Desc : This function is used to force repainting of an object. This may

be necessesary at certain times to achieve the desired result.

Use MainWindow (Case Sensitive) to repaint the Main Window.

Syntax : window paint windowname

window paint MainWindow

Example : window paint MainWindow

window paint Sys\_Win\_WIN

### WINDOW PaintFile

Desc : PaintFile is a special file that contains object painting commands which take place after any repainting (WINDOW PAINT). To paint objects on a screen open the PaintFile, then insert lines for each object you’d like to paint. After the lines have been entered into the PaintFile, issue a WINDOW PAINT command for the window you’d like to paint. Anytime that window receives a paint message (sizing the window, moving another window off of that window, etc…), the objects contained in the PaintFile will be drawn on the window. You can add objects anywhere in the PaintFile, or remove objects, and to force a refresh, issue the WINDOW PAINT command for that window. Each line in the PaintFile will contain the name of the window to paint to, type of the object, X position, Y position, Length, Height, Red value 0-255, Green value 0-255, Blue value 0-255, and misc. parameters depending on the object type. Each of the parameters are separated by a comma.

To paint on the Main Window, use “MainWindow” as the window name in lines within the PaintFile.

Object Types: (Note: Red = Red 0-255, Green=Green 0-255, Blue=Blue 0-255)

***Note: Use caution when forcing a paint of a window using WINDOW PAINT inside a***

***loop. Since a paint message is issued immediately and processing is multi-***

***threaded, the paint processing might be trying to use variables that are being***

***modified in the loop processing. To help prevent this problem you may want***

***to put any added/changed lines at the bottom of the PaintFile, and move to the top of the file prior to issuing a WINDOW PAINT command.***

**Text** Window name,Object name, Text,X position,

Y position,Length,Height,Red,Green,Blue,

Font Name,Text to Print (“Default” in the Font Name field will use the default font.

“Default” in the Font Name field will use the default font.

Example: TestWindow,Obj1,Text,100,100,15,15,0,0,0Times New Roman,Testing

**TextTilt** Rotated Text parameters: window name, Object name, TextTilt,X position,

Y position, Height,

Rotation Angle (0-360 degrees),Red,Green,Blue,Font Name,Text to Print

“Default” in the Font Name field will use the default font.

Example: TestWindow,Obj2,TextTilt,100,100,15,45,255,0,0,Default,45 red

**Line**  Line parameters: Window name, Object name, Line, Start X position,

Start Y position,

to X position, to Y position, Red, Green, Blue

Example: TestWindow,Obj3,Line,10,10,10,50,50,0,255,0

**Point**  Point parameters: Window name, Object name, Line, X position, Y position,

Red, Green, Blue

Example: TestWindow,Obj4,Point,10,10,50,0,0,255

**Rectangle** Hollow box parameters: Window name, Object name, Rectangle,

X position, Y position,

Length, Height, Red, Green, Blue

Example: TestWindow,Rectangle,10,10,50,50,0,255,0

**Box** Filled box parameters: Window name, Object name, Box, X position, Y position,

Length, Height, Red, Green, Blue

Example: TestWindow,Obj5,Box,10,10,50,50,0,255,0

**Box3D** Filled 3D box parameters: Window name, Object name, Box3D, X position,

Y position, Length, Height, Red, Green, Blue

Example: TestWindow,Obj6,Box3D,10,10,50,50,0,255,0

**BoxColor** Filled color box parameters: Window name, Object name, BoxColor, X position, Y position, Length, Height, Red, Green, Blue

Example: TestWindow,Obj6,BoxColor,10,10,50,50,0,255,0

**Circle** Hollow circle parameters: Window name, Object name, Circle, X position,

Y position, Length, Height, Red, Green, Blue

Example: TestWindow,Obj7,Circle,10,10,50,50,0,255,0

**Ellipse** Filled circle parameters: Window name, Object name, Ellipse, X position,

Y position, Length, Height, Red, Green, Blue

Example: TestWindow,Obj8,Ellipse,10,10,50,50,0,0,255

**Ellipse3D** Filled 3D circle parameters: Window name, Object name, Ellipse3D,

X position, Y position, Length, Height, Red, Green, Blue

Example: TestWindow,Obj9,Ellipse3D,10,10,50,50,0,0,255

**Diamond** Hollow diamond parameters: Window name, Object name, Diamond,

X position, Y position, Length, Height, Red, Green, Blue

Example: TestWindow,Objd,Diamond,10,10,50,50,255,255,0

**Bitmap** Draw a bitmap: Window name, Object name, Bitmap, X position,

Y position,Bitmap file Location

**BitmapInvert** Draw a bitmap with inverted colors: Window name, Object name,

Bitmap, X position, Y position,Bitmap file Location

**BitmapFit** Draw a bitmap fitting it into the defined size: Window name, Object name,

Bitmap, X position, Y position, Width, Height, Bitmap file location

**Note**: if Height is passed as zero, then Height will be calculated based on the

bitmap aspect ratio using the Width so the proportion of the bitmap is

maintained.

**BitmapFitInvert** Draw a bitmap fitting it into the defined size with inverted colors:

Window name, Object name, Bitmap, X position, Y position, Width, Height, Bitmap file location

**Note**: if Height is passed as zero, then Height will be calculated based on the

bitmap aspect ratio using the Width so the proportion of the bitmap is

maintained.

**Note**: for Bitmap objects if Height is passed as zero, then Height will be calculated based on the bitmap aspect ratio using the Width so the proportion of the bitmap is maintained.

**Arrow** Draw an arrow thin or wide optionally filled with a designated color:

Window name, Object name, Arrow1-53, X position, Y position, dx position, dy position, red, green, blue

**See Builder -> Windows Development –> Special Objects**

for a full list of all of the objects.

### WINDOW POINTER

Desc : This function is used to change the pointer to a different type.

It is good practice to change the pointer the the WAIT pointer

during long processes to let the user know that processing is

taking place. After the long process is complete change the pointer

back to the ARROW.

Syntax : window pointer type where type = arrow

pointer

wait

appicon

iconerror

iconinformation

iconquestion

iconwarning

move

sizenesw

sizens

sizenwse

sizewe

text

Example : window pointer wait

### WINDOW PUSH

Desc : This function is used to PUSH a button, Radio Button, or Check Box

as if the user clicked the mouse button while the pointer was over

the object. This is typically used when coding an ACTION for a list

box double click, you will want the double click to process as if

a button was pressed.

Note: This is a useful way to break up a long running process also. E.g. you might want

To show progress while processing files, and to allow painting to occur you

could set a re-enter variable, WINDOW PUSH a button to re-enter the process,

and at the start of the button process check to see if the re-enter variable has

been set, and if so then jump to the re-enter processing to continue where

processing left off when the WINDOW PUSH command was issued.

see WINDOW ACTION

Syntax : window push objname

Example : string Sys\_Win\_List\_LBOBJ = "window push Sys\_Win\_OK\_PBOBJ"

window push Sys\_Win\_Cancel\_PBOBJ

### WINDOW SELECT

Desc : This function is used to hilite a line in a list box.

You must set the list box string variable to "INDEX=n" where n is

the line that you want to hilite (1-max).

SEE WINDOW INSERT, WINDOW DELETE

Syntax : window select objname

Example : Sys\_Win\_List\_LBOBJ = "INDEX=1"

window select Sys\_Win\_List\_LBOBJ

### WINDOW SELECTACTION

Desc : This function is used by list box objects to set up a select

action. This works the same as WINDOW ACTION for a list box

double click, except that it works on a single click.

Do not use this function in conjunction with WINDOW ACTION.

SEE WINDOW ACTION

Syntax : window selectaction objname

Example : string Sys\_Win\_List\_LBOBJ = "call !a100\_Sys\_Win\_List\_LBOBJ"

window selectaction Sys\_Win\_List\_LBOBJ

### WINDOW SETFOCUS

Desc : This function is used to set the focus to a specific object on the

window. This is typically used when processing errors. When a field

has been found in error display an error message in a message box,

and then set the focus to the field that is in error. If the object

is a window then the focus will be moved into that window.

see WINDOW GETFOCUS, WINDOW GETFOCUSWIN

Syntax : window setfocus objname

Example : window setfocus Sys\_Win\_Entry1\_EFOBJ

### WINDOW SHADE

Desc : This function is used to pull the SHADE over the window to make it

invisible.

see WINDOW OPEN, WINDOW UNSHADE

Syntax : window shade winname

Example : window shade Sys\_Win\_WIN

### WINDOW SHOW

Desc : This function will make an object visible that was invisible by using

the HIDE function. You can use “window show MAIN” to show the main window after hiding it using WINDOW HIDE.

See WINDOW HIDE

Syntax : window show objname

Example : window show Sys\_Win\_Entry1\_EFOBJ

### WINDOW SIZE

Desc : This function allows you to resize any object that is open.

If you want to resize the main window then use 'MainWindow'

for the object name that you want to resize. You may use

MAX to size an object to it's maximum size.

Syntax : window size objname max

window size objname @maxvarstring

window size objname lowerleftpos, lowerright, width, height

window size objname @lowerleftpos, @lowerright, @width, @height

Example : window size MainWindow max

window size Sys\_Win\_Button1\_PBOBJ 10 10 50 50

window size Sys\_Win\_Button1\_PBOBJ @Xloc @Yloc @Xlen @Ylen

### WINDOW SOUND

Desc : This function generates sound from the speaker using the input

frequency (cycles-per-second 37-32767) and duration (milleseconds

1000 = 1 second).

Syntax : window sound # #

window sound @freq @dur

Example : window sound 300 100

window sound @Frequency @Duration

### WINDOW START

Desc : This function is used to begin PM processing and create the Main

window. All PM processing is done on this line. Any ACTION coded will

process on this line. If you have an error in the ACTION coded the

system will indicate that the error occurred on the WINDOW START line.

Based on the object that the action took place in find the string

variable that you assigned to the object's ACTION and check the code.

If you placed a CALL in the ACTION code processing will transfer to

the called subroutine and any error message line numbers will look

as you would expect. Do not place any other command on the

WINDOW START line. If ‘NOCLOSE’ follows WINDOW START, then all of the windows

presented in the system will not contain an ‘X’ in the upper right corner of the window which would allow the user to close the window using the ‘X’.

see WINDOW STOP, WindowInit, WindowWrap, WindowTimer

Syntax : window start

Window start noclose

Example : window start

### WINDOW STOP

Desc : This function is used to stop PM processing the same as if you double

click on the main window system menu. This is typically used when the

user selects an EXIT selection, or when a crittical error occurs

in your processing. The EXIT function acts the same as a

WINDOW STOP.

see WINDOW START

Syntax : window stop

Example : window stop

### WINDOW TIMER

Desc : This function is used to start the timer. You define the interval and

when the interval takes place the code assigned to the string variable

WindowTimer will take place. Create the STRING variable WindowTimer

and assign the action code to it before starting the timer. The interval

may be 1/10 seconds, or 1 to 60 seconds. This is typically used to

blink fields or to display the time on the main window title. To

turn off the timer use WINDOW TIMER OFF.

Syntax : window timer on n (where n = 1-60 sec, 0=1/10 sec)

window timer off

Example : string WindowTimer = "call !a100\_WindowTimer"

window timer on 1

...

!a100\_Window\_Timer

Title = "Main Window @date@ @time@"

window title MainWindow

return

### WINDOW TITLE

Desc : This function allows you to change the title on any window. If the

object name is 'MainWindow' then the title of the Main Window will

be changed. Assign the title text to the string variable before

processing the TITLE function.

Syntax : window title objname

Example : MainWindow = "Main Window Title"

window title MainWindow

Sys\_Win\_WIN = "Secondary Window Title"

window title Sys\_Win\_WIN

### WINDOW UNCHECK

Desc : This function is used to put a un-check mark next to a menu

selection.

See WINDOW CHECK.

Syntax : window uncheck msobj

Example : window uncheck Sys\_Main\_Selection\_MSOBJ

### WINDOW UNCONCEAL

Desc : This function is used to UNCONCEAL an entry field after it has

been CONCEALed. You Must UNCONCEAL an entry field just before it's

window is OPENed!

see WINDOW CONCEAL.

Syntax : window unconceal Sys\_Win\_Entry\_EFOBJ

Example : window unconceal Sys\_Win\_Password\_EFOBJ

window open Sys\_Passwordwin\_WIN

### WINDOW UNSELECT objname

Desc : This function is used to un-hilite a line in a list box.

You must set the list box string variable to "INDEX=n" where n is

the line that you want to un-hilite (1-max).

s ee WINDOW SELECT, WINDOW INSERT, WINDOW DELETE

Syntax : window unselect objname

Example : S ys\_Win\_List\_LBOBJ = "INDEX"

window obtain Sys\_Win\_List\_LBOBJ

Sys\_Win\_List\_LBOBJ = "INDEX=@Sys\_Win\_List\_LBOBJ@"

window unselect Sys\_Win\_List\_LBOBJ

### WINDOW UNSHADE

Desc : This function is used to lift the SHADE off the window to make it

visible.

see WINDOW OPEN, WINDOW SHADE

Syntax : window unshade winname

Example : window unshade Sys\_Win\_WIN

### WINDOW WINDOWPOS

Desc : This function is used to retrieve an object's location and

size.

See WindowDown1, WindowDown2, WindowDown3, WindowUp1, WindowUp2,

WindowUp3, WindowMouse, WINDOW SIZE

Syntax : window windowpos objectname Xpos Ypos Width Height

Example : window windowpos Sys\_Win\_Test\_EFOBJ XPos YPos Width Height

### WRITE

Desc : Write the contents of an open file to a filestring. After the write the pointer is placed at the top of the file. If the filename is specified as \* then the current file will be written. If the filestring is \* then the file will be written to the current filestring.

See OPEN, READ, CLOSE.

Syntax : write name filestring

write @name@ @filestring@

write \* filestring

write \* @filestring

write \* \*

Example : write filea c:\bedit\bedit.dat